

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V  
BRATISLAVE  
STROJNÍCKA FAKULTA**

**MOTO SHIELD: MINIATÚRNE DIDAKTICKÉ  
ZARIADENIE PRE RIADENIE JEDNOSMERNÉHO  
MOTORA**

**Diplomová práca**

**SjF-5226-92875**

Študijný program: Automatizácia strojov a procesov  
Študijný odbor: kybernetika  
Školiace pracovisko: Ústav automatizácie, merania a aplikovanej informatiky  
Vedúci záverečnej práce: doc. Ing. Martin Gulan, PhD.  
Konzultant: Ing. Erik Mikuláš

**Bratislava, 2022**

**Bc. Ján Boldocký**



## ZADANIE DIPLOMOVEJ PRÁCE

Študent: **Bc. Ján Boldocký**  
ID študenta: 92875  
Študijný program: automatizácia a informatizácia strojov a procesov  
Študijný odbor: kybernetika  
Vedúci práce: doc. Ing. Martin Gulan, PhD.  
Vedúci pracoviska: doc. Ing. Martin Halaj, PhD.  
Konzultant: Ing. Erik Mikuláš

Názov práce: **MotoShield: Miniatúrne didaktické zariadenie pre riadenie jednosmerného motora**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Zadanie práce možno zhrnúť do nasledujúcich bodov:

1. zhodnotenie súčasného stavu zariadenia MotoShield, návrh vylepšenej schémy zapojenia, elektronických komponentov, a dosky plošných spojov;
2. vývoj programátorského rozhrania (API) na ovládanie navrhnutého zariadenia pre rôzne programovacie jazyky a vývojové prostredia (C/C++ pre Arduino IDE, MATLAB, Simulink, príp. iné) s využitím prostredia GitHub na manažovanie verzií softvéru a integráciu do základnej knižnice AutomationShield.;
3. formulácia didaktických úloh a inštrukčných príkladov pre metódy spätnoväzbového riadenia vo vytvorených API;
4. analýza matematicko-fyzikálneho modelu, návrh experimentu a identifikácia neznámych parametrov modelu prostredníctvom nameraných údajov;
5. vývoj aplikácie na zjednodušené spustenie experimentov so zvolenými parametrami a grafické zobrazenie priebehu vybraných veličín.

Rozsah práce: 60 – 80 strán

Termín odovzdania diplomovej práce: 27. 05. 2022

Dátum schválenia zadania diplomovej práce: 13. 05. 2022

Zadanie diplomovej práce schválil: prof. Ing. Cyril Belavý, CSc. – garant študijného programu

Predovšetkým by som sa chcel podakovať vedúcemu práce, doc. Ing. Martinovi Guilanovi, PhD., za odbornú pomoc, trpežlivosť ako aj usmernenie a pri vypracovaní diplomovej práce. Ospravedlňujem sa za nepríjemnosti v zmysle oneskorenia a nesmierne som vďačný za Váš optimizmus pri odovzdávaní. Taktiež by som sa chcel podakovať aj konzultantom, Ing. Erikovi Mikulášovi a Mgr. Ing. Anne Vargovej za radu pri riešení danej problematiky a za upokojujúce rozhovory.

Taktiež sa chcem ešte raz podakovať Prof. Ing. Gergelyovi Takácsovi PhD. za usmernenie počas štúdia, vedomosti a rady, ktoré som od Vás dostal. Lúto mi je, že v posledné dni štúdia sa naše cesty rozdelili.

Taktiež by som sa chcel podakovať aj všetkým spolužiakom za spríjemnenie študentského života. Prevažne Bc. Denisovi Skokanovi a Bc. Dávidovi Kožuškaničovi, ktorí boli pri mne od začiatku štúdia. Ďakujem Vám za všetky pekné chvíle počas štúdia, za duševnú podporu v ťažkých momentoch a za nepretržité päť ročné kamarátstvo. Už verím, že nás nespája iba škola a želám aby naše kamarátstvo ešte dlho zotrvalo. Zvlášť by som sa chcel podakovať Bc. Denisovi Skokanovi, že ma srdečne privítal vo svojej krajinе a že ma nikdy nepovažoval za cudzinca. Za gramatické korekcie a podporu pri vypracovaní diplomovej práce sa chcem podakovať Jakubovi Hnilicovi.

Bratislava, 25. mája 2022

Bc. Ján Boldocký

**Názov práce:** MotoShield: Minitáürne didaktické zariadenie pre riadenie jednosmerného motora

**Kľúčové slová:** MotoShield, AutomationShield, jednosmerný motor, riadenie, Arduino, identifikácia, grafické užívateľské rozhranie

**Abstrakt:** Práca je zameraná na ďalší rozvoj didaktického zariadenia MotoShield v rámci open-source iniciatívy AutomationShield. MotoShield je miniaturizované zariadenie, navrhnuté primárne na výučbu kybernetiky a na riešenie problematiky ohľadom riadenia jednosmerných motorov. Zariadenie je kompatibilné s prototypizačnými doskami rodiny Arduino a s doskami od rôznych iných výrobcov. Cenová dostupnosť zariadenia je neporovnatelná s kommerčnými didaktickými pomôckami rovnakého typu. Toto je výhodou, medzi iným, aj pri dištančnej forme štúdia, pretože by každý študent mohol mať zaobstaraný prístroj na získavanie praktických skúseností. Na začiatku práce je popísaný východiskový stav prístroja MotoShield z hľadiska hardvérového vyhotovenia a softvérovej podpory, kde autor popisuje aj hlavné nedostatky. Ďalej je popísaný návrh vylepšenej verzie hardvéru a softvéru. V rámci softvérovej podpory sú dostupné programátorské rozhrania pre programovacie jazyky C/C++, MATLAB a pre prostredie Simulink, ktorých účelom je zjednodušiť ovládanie prístroja. V pokračovaní sú popísané ukážkové príklady vrátane matematicko-fyzikálnej analýzy, identifikácie a spätnoväzbového riadenia. V poslednej kapitole je popísaný postup vývoja grafického rozhrania na jednoduché spustenie a sledovanie vytvorených príkladov.

**Title:** MotoShield: a Miniature Didactic Device for DC Motor Control

**Keywords:** MotoShield, AutomationShield, DC Motor, System Control, Arduino, System Identification, Graphical User Interface

**Abstract:** This master's thesis is focused on the further development of the inexpensive MotoShield didactic device in connection to the AutomationShield open-source initiative. MotoShield is a miniaturized device primarily designed for education in the field of control engineering specifically, concerning the DC motor control. The proposed device is compatible with the microcontroller boards from the Arduino project and with boards from various other manufacturers. The price of the device is incomparable to its commercial counterparts, which is convenient especially when it comes to remote teaching. Due to the fact that nearly every student could have access to such device and thus gain the hands-on experience. In the introductory chapter, the author delineates the initial hardware and software state of the device and also underlines its main disadvantages. Subsequently, the author outlines a more optimized hardware blueprint and an improved programming interface for programming languages such as C/C++, MATLAB and, Simulink. The idea behind programming interface is to simplify the use of the device, in other words, abstracting the programming on a hardware level. The following chapters consist of demonstrational examples for mathematical-physical analyses, system identification and feedback control. In the final chapter, the author delves into the development of the graphical user interface that ultimately enables the user to experiment with the device with minimal efforts.

# Obsah

**Zoznam obrázkov**

**Zoznam skratiek**

**Zoznam matematických symbolov**

<b>Úvod</b>	<b>1</b>
<b>1 Východiskový stav</b>	<b>4</b>
1.1 Hardvér - MotoShield R1 . . . . .	4
1.2 Softvér - MotoShield R1 . . . . .	6
1.3 Zhrnutie nedostatkov . . . . .	9
<b>2 Návrh vylepšenej verzie hardvéru</b>	<b>10</b>
2.1 Návrh elektronických komponentov . . . . .	10
2.1.1 Pohon motora . . . . .	10
2.1.2 Meranie prúdu . . . . .	11
2.2 Návrh schémy zapojenia . . . . .	12
2.3 Návrh dosky plošných spojov . . . . .	14
2.4 Cena . . . . .	15
<b>3 Vývoj programátorského rozhrania pre MotoShield R2</b>	<b>16</b>
3.1 Riešenie konfliktov API pre MotoShield R1 . . . . .	16
3.2 API pre prostredie Arduino IDE . . . . .	17
3.2.1 Konštruktor . . . . .	18
3.2.2 Metódy na ovládanie motora . . . . .	19
3.2.3 Metódy na meranie výstupných veličín . . . . .	20
3.2.4 Pomocné metódy . . . . .	21
3.3 API pre prostredie MATLAB . . . . .	24
3.4 API pre prostredie Simulink . . . . .	27
3.4.1 Blok Reference Read . . . . .	28
3.4.2 Blok Actuator Write . . . . .	29
3.4.3 Blok Sensor Read Current . . . . .	29
3.4.4 Blok Sensor Read . . . . .	30
3.4.5 Blok MotoShield . . . . .	31

<b>4 Modelovanie, identifikácia a riadenie systému</b>	<b>33</b>
4.1 Matematicko-fyzikálna analýza . . . . .	33
4.2 Identifikácia a odhad neznámych parametrov . . . . .	35
4.3 Odhad stavu . . . . .	38
4.4 Spätnoväzbové riadenie . . . . .	39
4.4.1 Kompenzátor . . . . .	39
4.4.2 PID riadenie - metóda Ziegler-Nichols . . . . .	42
4.4.3 Stavové riadenie . . . . .	44
4.5 Didaktické úlohy . . . . .	49
4.5.1 Modelovanie . . . . .	50
4.5.2 Identifikácia modelu . . . . .	52
4.5.3 Riadenie . . . . .	54
<b>5 Grafické užívateľské rozhranie</b>	<b>57</b>
<b>6 Záver</b>	<b>62</b>
<b>Literatúra</b>	<b>63</b>

# Zoznam obrázkov

1	Prístroje z projektu Automationshield pripojené na Arduino UNO [9]	2
1.1	MotoShield R1 pripojený na Arduino UNO (prebraté z [11]). . . . .	4
1.2	Výstupné signály z Hallovho snímača. . . . .	5
1.3	Integrovaný obvod L293D a pravdivostná tabuľka. . . . .	6
1.4	Nameraný signál odberu prúdu - MotoShield R1 (prebraté z [4]). . . . .	7
1.5	El. schéma zapojenia prístroja MotoShield R1 (prebraté z [11]). . . . .	8
2.1	Integrovaný obvod ZXBM5210 a pravdivostná tabuľka. . . . .	10
2.2	Integrovaný obvod INA169. . . . .	11
2.3	Schéma zapojenia el. obvdou prístroja MotoShield R2. . . . .	12
2.4	Doska plošných spojov - MotoShield R2. . . . .	14
2.5	MotoShield R2 pripojený na Arduino UNO. . . . .	15
3.1	Sada blokov Simulink API pre prístroj Motoshield. . . . .	27
3.2	Vnútorná štruktúra bloku Reference Read. . . . .	28
3.3	Vnútorná štruktúra subsystému <b>ActuatorWriteR2</b> , bloku Actuator Write. . . . .	30
3.4	Vnútorná štruktúra subsystému <b>SensorReadR2</b> , bloku Sensor Read. . . . .	31
3.5	Blok MotoShield zapojený v regolačnom obvode (prebraté z [4]). . . . .	32
4.1	Viac hladinový pseudonáhodný testovací signál. . . . .	36
4.2	Porovnanie filtrovaného a nameraného signálu. . . . .	37
4.3	Porovnanie modelu prenosovej funkcie s dátami. . . . .	38
4.4	Porovnanie modelu stavového modelu s dátami. . . . .	39
4.5	Porovnanie výstupu a odhadu stavu na základe identifikovaného prenosu. . . . .	40
4.6	Prechodová charakteristika prenosovej funkcie. . . . .	41
4.7	Schéma riadiaceho obvodu „Lag“ kompenzátoru. . . . .	42
4.8	Regulačný pochod pri riadení pomocou „Lag“ kompenzátoru. . . . .	43
4.9	PID riadenie - Ziegler-Nicholsova metóda ladenia. . . . .	44
4.10	LQ riadenie bez integračnej zložky a s integračnou zložkou. . . . .	48
4.11	Viac hladinový signál žiadanej hodnoty. . . . .	56
4.12	Sínusový signál žiadanej hodnoty. . . . .	56
5.1	Grafické užívateľské rozhranie. . . . .	58
5.2	PID riadenie cez grafické rozhranie. . . . .	60

# Zoznam skratiek

**AD, ADC** Analógovo-digitálny (prevodník)

**API** Programátorské rozhranie (angl. Application Programming Interface)

**CMOS** (angl. Complementary Metal Oxide Semiconductor)

**ISR** Obsluha prerušenia (angl. Interrupt Service Routine)

**PCB** Tlačená obvodová doska (angl. Printed circuit board))

**PID** Proporcionálne integračne derivačný (regulátor) (angl. Proportional–integral–derivative)

**PPR** Počet pulzov na otáčku (angl. Pulses Per Revolution)

**PWM** Šírkovo modulovaný (signál) (angl. Pulse-width Modulation)

**SI** Medzinárodná sústava jednotiek (franc. Système international)

**SMT** Technológia povrchovej montáže (angl. Surface-mount technology)

**THT** Technológia osadzovanej montáže (angl. Through-hole technology)

**CI** Priebežná integrácia (angl. Continuous Integration)

**PCB** Doska plošných spojov (angl. Printed Circuit Board)

**LED** Light-emitting diode

**LQ** Lineárne kvadratický (angl. Linear Quadratic)

**RegEx** Regulárny výraz (angl. Regular expression)

**CLI** Príkazový riadok (angl. Command-Line Interface)

# Zoznam matematických symbolov

Symbol	Veličina	Jednotka (použitá v práci)
$b$	Koeficient viskózneho tlmenia	Nms
$G(s)$	Prenosová funkcia	-
$I(s)$	Elektrický prúd (obrazová oblast)	A
$i$	Elektrický prúd	A
$J$	Moment zotrvačnosti	kg·m <sup>2</sup>
$K_e$	Koeficient elektromotorickej sily	Vrad <sup>-1</sup> s <sup>-1</sup>
$K_p$	Proporcionálna konštantá	-
$K_t$	Koeficient momentu motora	NmA <sup>-1</sup>
$k$	Index vzorky	-
$kT$	Diskrétny čas	s
$L$	Indukčnosť	H
$M$	Krútiaci moment	Nm
$M_h$	Hnací moment	Nm
$M_t$	Trecí moment	Nm
$R$	Elektrický odpor	$\Omega$
$r$	Referenčná hodnota	-
$s$	Komplexná premenná	-
$t$	Spojity čas	s
$T_d$	Derivačná časová konštantá	-
$T_i$	Integračná časová konštantá	-
$t_0$	Doba aktívneho stavu	-
$t_p$	Periódna PWM signálu	-

$T_s$	Periódna vzorky	s
$u$	Akčný zásah	-
$U_{\text{RMS}}$	Efektívna hodnota napäťia	V
$U_z$	Vstupné napätie	V
$V$	Amplitúda PWM signálu	V
$V_{\text{OUT}}$	Výstupné napätie senzoru prúdu	V
$x$	Stavová veličina	-
$y$	Výstupná veličina	-
$\mathcal{E}$	Elektromotorická sila	V
$\theta$	Uhol otočenia	rad
$\dot{\theta}$	Uhlová rýchlosť	rad·s <sup>-1</sup>
$\ddot{\theta}$	Uhlové zrýchlenie	rad·s <sup>-2</sup>

# Úvod

V oblasti technických vied je výučba založená na teoretických vedomostiach o javoch v prírode a ich matematicko-fyzikálnej analýze. Keďže prírodné javy v reálnych systémoch nie sме schopný popísať v úplnosti, sме nútene overiť správnosť matematickej analýzy porovnaním správania sa matematického modelu a reálneho systému.

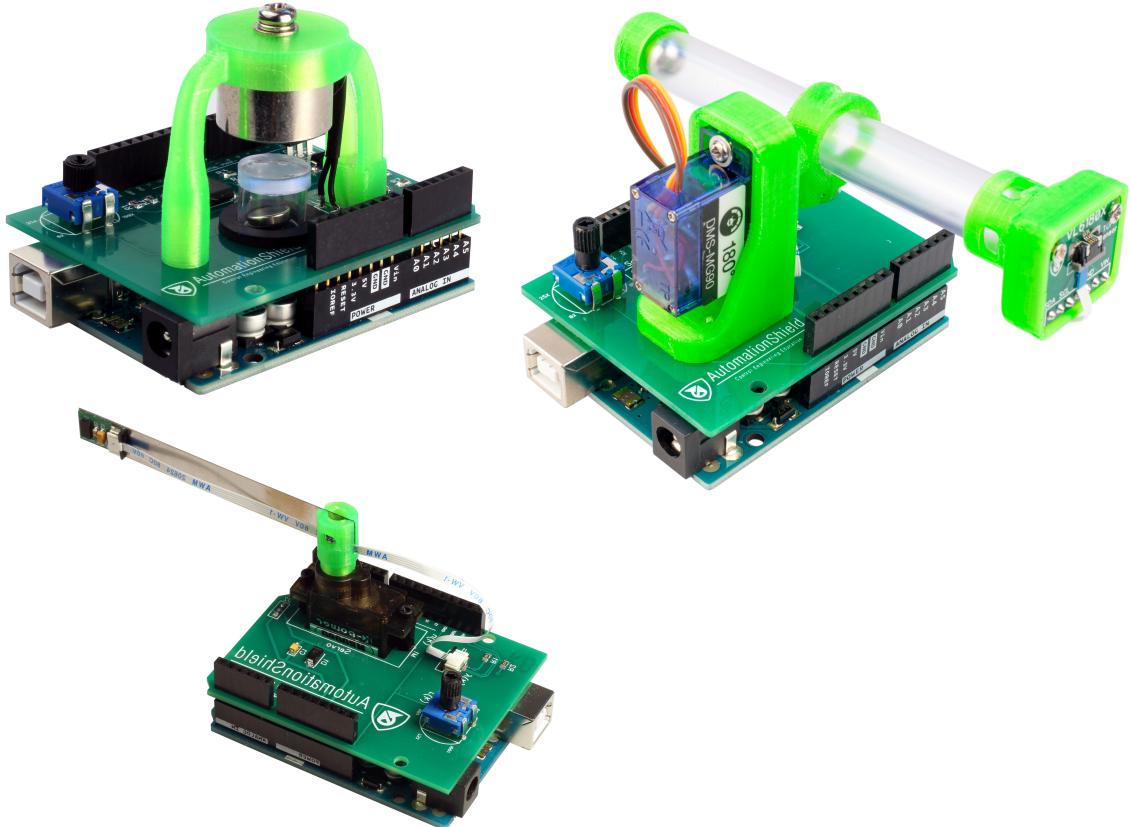
Takúto metódu overenia teoretického popisu nie sме vždy schopný predvíeť, väčšinou kvôli náročnosti na fyzické zostavenie systémov či z hľadiska konštruovania, technológií alebo financií. V akademickom svete je najčastejším problémom práve nedostatok finančných prostriedkov a preto sa na Ústave *Automatizácie, Merania a Aplikovanej informatiky Strojníckej fakulty STU* zrodila iniciatíva *AutomationShield*. V rámci iniciatívy sa študenti pod vedením učiteľov zaoberajú vývojom jednotlivých prístrojov na vykonávanie experimentov a výučbu automatizačnej techniky. Prístroje predstavujú miniaturizované technické systémy s rôznorodou problematikou z hľadiska popisu prírodných javov ako napr.:

- LinkShield - Aktívne tlmenie vibrácií pružného nosníka pomocou servo-motora,
- BoBShield - Riadenie polohy guličky v uzavretej trubici za pomocí naklonenia okolo horizontálnej osi,
- OptoShield - Riadenie svietivosti svetelnej diódy za pomocí fotorezistora,
- FloatShield - Riadenie polohy vznášajúcej sa lopty v trubici za pomocí ventilátora,
- BlowShield - Riadenie tlaku v uzavretej nádobe,
- MotoShield - Riadenie jednosmerného motora s komutátorom za pomocí inkrementálneho tachometra.

Prínosom iniciatívy *AutomationShield* pre celosvetovú akadémiu je, že jednotlivé prístroje vyžadujú minimálne finančné náklady na vyhotovenie, ich kompletná technická dokumentácia je verejne dostupná a ku každému prístroju prislúcha aj programátorské rozhranie pre rôzne programovacie jazyky na jednoduché ovládanie a rôzne ukážkové príklady. Prístroje sú relatívne malých rozmerov a preto sú vhodné aj na zadania, kedy si študent daný prístroj požičia, čo môže byť veľkou výhodou pri diaľkovej forme štúdia.

Uvedené prístroje sú navrhnuté ako moduly pre rôzne prototypizačné dosky so štandardizovaným rozstupom vstupno-výstupných hlavíc. Čiže sú kompatibilne s doskami ako napr. *Arduino UNO R3* (Obr. 1), *Arduino Mega 2560 R3*, *Arduino*

*Due, Adafruit Metro M4, STM32F103* a podobne, čo umožňuje flexibilitu, resp. široký výber pri voľbe mikroradičových dosiek.



Obr. 1: Prístroje z projektu Automationshield pripojené na Arduino UNO [9]

Pri vývoji uvedených didaktických prístrojov sú vo väčšine prípadov používané prototypizačné mikroradičové dosky *Arduino*. *Arduino* je, podobne ako aj *AutomationShield*, projekt zrodený na akademickej pôde, ktorého cieľom je poskytnúť nízkonákladové pomôcky pre študentov a začiatočníkov v oblasti elektronike. Projekt *Arduino* obsahuje dôkladnú a verejne dostupnú dokumentáciu, jeho popularita resp. komunita neustále rastie a preto je podpora na fórach bohatá. Taktiež bolo vyvinuté aj programátorské prostredie *Arduino IDE* a rozhranie na zjednodušené programovanie mikroradičov. Dosky *Arduino* je možné programovať aj v prostredia MATLAB a Simulink, keďže existujú oficiálne balíky od zamestnancov firmy *Math Works*.

Zameraním práce je pokračovať vo vývoji didaktickej pomôcky **MotoShield**. Ako bolo hore uvedené, je to prístroj, pri ktorom je hlavná problematika riadenie jednosmerného motora s komutátorom pomocou inkrementálneho snímača a snímača odberu prúdu. Podobné projekty a komerčné zariadenia existujú, avšak každý z nich má nedostatky, ktorím sa pri MotoShielde snažíme vyhnúť, Tab. (2).

Tabuľka 2: Porovnanie didaktických zariadení na riadenie jednosmerného motora.

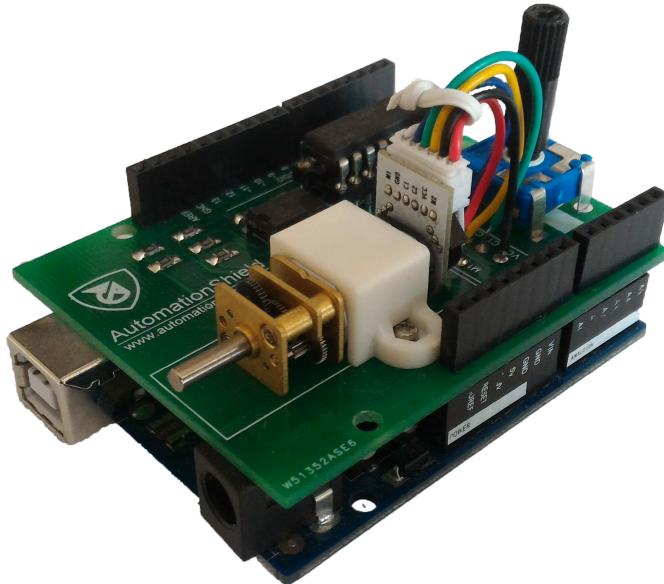
Názov	Programátorské rozhranie	Licencia	Rozmer	Cena
Quanser QNET DC Motor Board 2.0	LabView	Komerčná	Kompaktný	1850 €
FESTO Training System with DC Motor	-	Komerčná	Nekompaktný	-
D.C. Motor Control Using NI myRIO-1900 [13]	LabView	Open Source	Kompaktný	600 €
Didactic platform for DC motor speed [5]	Nemá	Open Source	Kompaktný	182 €
MotoShield	Arduino IDE (C/C++) MATLAB Simulink	Open Source	Kompaktný	19.92 €

V rámci práce sa budeme snažiť:

- identifikovať nedostatky prvej verzie prístroja MotoShield a zároveň ich kompenzovať pri vývoji druhej verzie. V rámci projektu sú prístroje takisto metodikou iteratívne vylepšované.
- Zmena hardvérovej zostavy si vyžaduje prispôsobenie programátorských rozhraní. Rozhrania nebude písat od začiatku, ale budeme stavať na tom, čo už máme, pričom vytvoríme možnosť voľby verzie prístroja.
- Na overenie funkčnosti hardvérovej zostavy, v symbioze s programátorskými rozhraniami, vytvoríme príklady zamerané prevažne na riadenie.
- Keďže sa snažíme vytvoriť didaktický prístroj, sformulujeme aj príklady týkajúce sa kybernetiky a mechatroniky. Študenti môžu definované úlohy samostatne vypracovať alebo sa s úlohou stretnú v podobe zadania.
- Na koniec sa posnažíme vytvoriť grafické užívateľské rozhranie, ktoré je atrakciou hlavne pre začiatočníkov. Vývoj grafického rozhrania môže byť komplexný a preto sa budeme snažiť zachovať škálovateľnosť a prehľadnosť kódu s cieľom, aby vývojári mohli pracovať na problematike spojenými silami.

# 1 Východiskový stav

Prístroj MotoShield bol prvýkrát spomenutý v bakalárskej práci Tibora Konkolya [11], kde študent navrhol koncept a zároveň aj prvú verziu hardvérovej realizácie (Obr. 1.1). Základná idea konceptu bola vytvoriť prístroj, ktorý bude obsahovať jednosmerný motor, snímač na meranie odberu prúdu a snímač na meranie uhlovej rýchlosťi hriadeľa. Keďže je prístroj v konečnom dôsledku navrhovaný na výučbu automatizačnej techniky, resp. riadenia, podmienkou bolo navrhnúť aj programátorské rozhranie na to, aby študenti už nemuseli písat softvér na nízkej úrovni, teda na spojazdnenie hardvéru. Programátorské rozhranie umožní študentom jednoduchým spôsobom ovládať vstupy, výstupy, nastavenie vzorkovacej periódy a podobne.

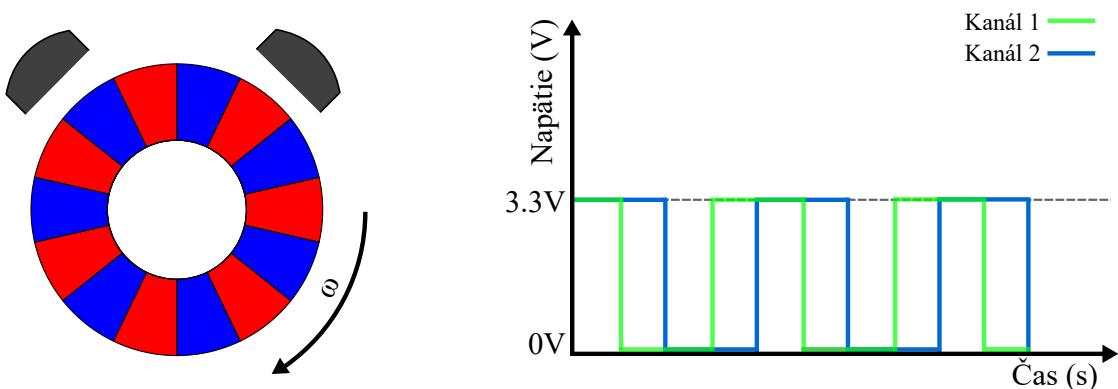


Obr. 1.1: MotoShield R1 pripojený na Arduino UNO (prebraté z [11]).

## 1.1 Hardvér - MotoShield R1

Úplná prvá verzia hardvéru bola navrhnutá v bakalárskej práci Tibora Konkolya. Študent navrhol použiť jednosmerný motor s komutátorom a prevodovkou na prednom hriadieli s pomerom 380:1 od firmy DFRobot [7]. Tento motor má na zadnom hriadieli pripojené magnetické koliesko so siedmymi párami pólov. Na detegovanie rotačného pohybu kolieska je zo zadnej strany nasadená doska plošných spojov,

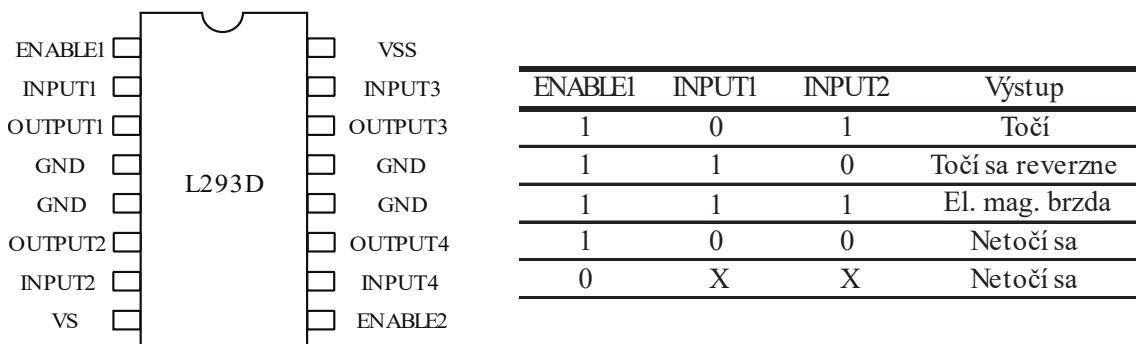
ktorá zahŕňa dve Hallove sondy a príslušné odpory. Vstupno-výstupných hlavíc na motore je šesť, z ktorých M1 a M2 sú pripojené na svorky motora, hlavice VCC a GND sú na napájanie Hallovho snímača. Posledné dve hlavice, C1 a C2 sú výstupy z Hallových sond. Výstupom sú impulzné signály s dolnou hodnotou 0V a hornou hodnotou rovnakou ako napájanie VCC. Signály z Hallových sônd sú mikroradiču privodené na digitálne vstupy D3 a D4 Obr. 1.5, čo nie je správne, keďže tieto signály na meranie uhlovej rýchlosť vyžadujú spúštanie inštrukcií prostredníctvom externého prerušenia. Pri mikroradičoch ATmega382P a ATmega2560 vstup D4 nie je vhodný na použitie externého prerušenia a preto merat' uhlovú rýchlosť je možné iba pomocou vstupu D3, čiže iba pomocou jednej sondy Hallovho snímača.



Obr. 1.2: Výstupné signály z Hallovho snímača.

Ked'že výrobca uviedol, že odber prúdu motora je väčší ako 20mA, nie je odporúčané napájať motor priamo z digitálneho výstupu mikroradiča. Na pohon motora bol navrhnutý H-mostík v podobe integrovaného obvodu L293D v prevedení PDIP(16), pomocou ktorého sme schopní ovládať smer otáčania motora. Navrhnutý integrovaný obvod obsahuje dva kanály a keďže ovládame jeden motor, použitý je len jeden kanál a taktiež obvod obsahuje aj diódy na elimináciu indukovaného spätného prúdu. Na ovládanie smeru otáčania sú použité vývody INPUT1 a INPUT2 podľa pravdivostnej tabuľky na Obr. (1.3), na napájanie vnútornej logiky integrovaného obvodu je pin VCC a motor je napájaný cez vývod VM.

Meranie prúdu bolo realizované pomocou meracieho odporu  $R$  a príslušnej periférie, viď Obr. 1.5. Princíp je založený na tom, že prúd je priamoúmerný úbytku napätia na odpore. Úbytok napätia je získaný rozdielom napätí pred a za odporem. Odčítanie týchto signálov je realizovaný pomocou prvého kanálu operačného zosilňovača LM358. Druhý kanál je vyhradený na zosilnenie, resp. násobenie rozdielu napätia podľa pomeru odporov  $R_f$  a  $R_g$ . V navrhnutom riešení je faktor zosilnenia 2.96 [11]. Takýto zosilnený signál je privodený mikroradiču cez analógový vstup A3. Roztočením motora pri rôznom akčnom zásahu je meraný signál zašumený a maximálna amplitúda je 16 v meracom rozsahu 0 až 1023. Na meracom rozsahu 10-bitového analógovo-digitálneho prevodníka je takáto amplitúda nízka a preto prichádza k nepresnostiam. Preto v nasledovnej verzie zariadenia bolo navrhnuté nové riešenie na meranie prúdu.



Obr. 1.3: Integrovaný obvod L293D a pravdivostná tabuľka.

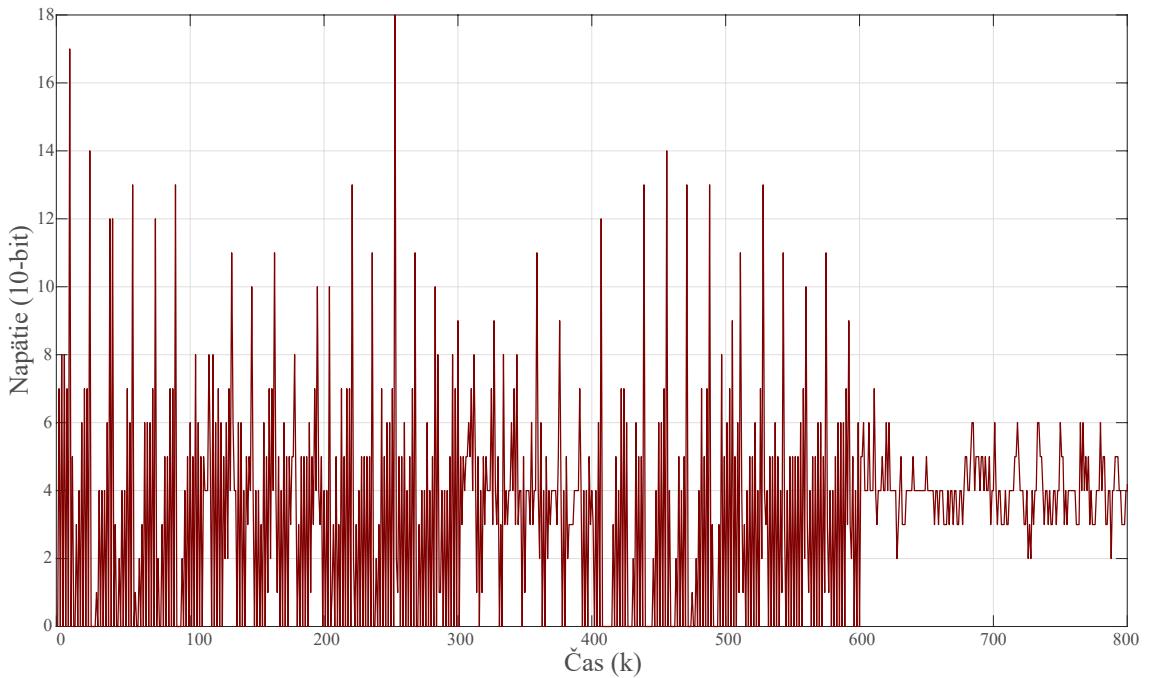
## 1.2 Softvér - MotoShield R1

Pri návrhu prvej verzie hardvéru bolo navrhnuté aj programátorské rozhranie (angl. Application Programming Interface) [11]. Avšak toto rozhranie obsahovalo početné nedostatky a preto v rámci bakalárskej práce *MotoShield: Vývoj programátorského rozhrania, identifikácia, riadenie a úprava didaktického prístroja na riadenie rýchlosťi motorov*[4] bolo napísané rozhranie podľa pravidiel projektu *AutomationShield*[9]. Rozhrania boli napísané v jazyku C++ pre prostredie Arduino IDE a taktiež aj pre prostredia MATLAB a Simulink.

Pre každé prostredie boli napísané základné metódy, resp. bloky na ovládanie vstupu a výstupov. V jazyku C++ a MATLAB boli API napísané formou objektovo orientovaného programovania, kde trieda `MotoShieldClass`, resp. `MotoShield` v prípade MATLAB-u zahrňa všetky metódy a premenné programátorského rozhrania. Ovládanie vstupu, resp. zápis akčného zásahu je riadenie pomocou metódy `actuatorWrite(float)`, ktorá ako vstupný argument očakáva hodnotu 0 až 100 (percent). Na ovládanie smeru otáčania bola napísaná metóda `setDirection(bool)`. Táto metóda pomocou digitálnych výstupov mikroradiča D6 a D7 nastaví integrovaný H-mostík podľa pravdivostnej tabuľke na Obr. 1.3. Ďalej bola napísaná aj metóda `actuatorWriteVolt(float)`, kde vstupný argument je hodnota na pohon motora uvedená v jednotke Volt. Kedže prototypizačné dosky Arduino UNO a Arduino Mega 2560 neobsahujú digitálno-analógový prevodník, el. energiu na motor vieme dávkovať iba pomocou pulzno šírkovej modulácie (angl. Pulse Width Modulation<sup>1</sup>) číslicového signálu s amplitúdou 5V. Prevod, resp. approximáciu činiteľa plnenia modulácie číslicového signálu na základnú jednotku napäťia umožňuje vzťah kvadratického priemeru (angl. Root Mean Square<sup>2</sup>), Rov. (1.1), kde  $t_0$  predstavuje hodnotu činiteľa plnenia,  $t_p$  je maximálna hodnota činiteľa plnenia a  $V$  je amplitúda PWM signálu. Symbol  $U_{RMS}$  predstavuje approximovanú hodnotu napäťia [17].

<sup>1</sup>abbrev. PWM.

<sup>2</sup>abbrev. RMS.



Obr. 1.4: Nameraný signál odberu prúdu - MotoShield R1 (prebraté z [4]).

$$U_{\text{RMS}}(t) = V \sqrt{\frac{t_0(t)}{t_p}} \implies t_0(t) = \frac{U_{\text{RMS}}^2(t)}{V^2} t_p. \quad (1.1)$$

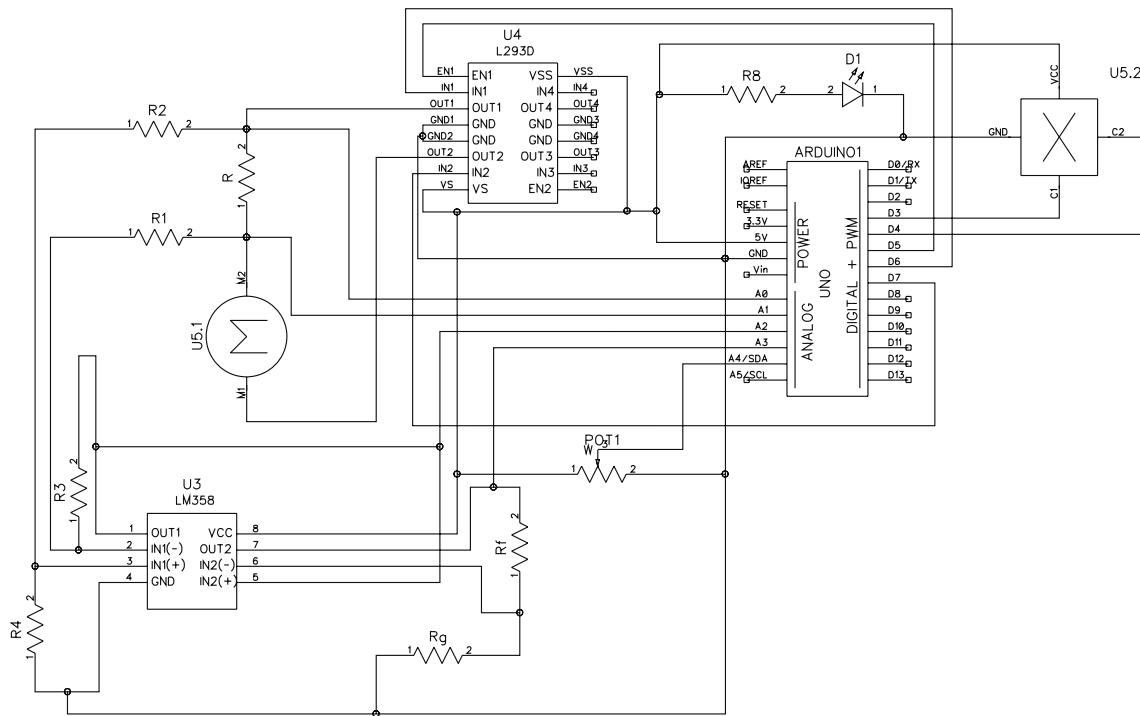
Podľa pokynov na návrh prístrojov v rámci projektu *AutomationShield* je dané, že každý prístroj by mal obsahovať potenciometer na ručné nastavenie referenčnej, resp. žiadanej hodnoty pri riadení. Na vyčítanie polohy potenciometra bola napísaná metóda `referenceRead`, ktorá vráti hodnotu 0 až 100 percent.

Pre meranie prúdu je metóda pomenovaná `sensorReadCurrent`, ktorá vráti hodnotu prúdu v jednotke mA. V rámci metódy sa vyčíta hodnota z analógovo-digitálneho prevodníka pomocou inštrukcie `analogRead`, ktorá pôvodí zo štandardnej knižnice Arduino. Vrátená hodnota je iba preškálovaná vzhľadom na zosilnenie neinvertujúceho operačného zosilňovača a meracieho odporu.

Na meranie uhlovej rýchlosťi je určená metóda `sensorReadRPMPerc`. Na vysvetlenie tejto metódy si najprv musíme vysvetliť inicializačnú metódu `begin` (`float`) a kalibračnú metódu `calibration`.

Pri použití tohto programátorského rozhrania je žiadané inicializačnú metódu zavolať ako prvú. Táto zabezpečí správne nastavenie vstupov a výstupov. Ďalej, inicializačná metóda zabezpečí aj nastavenie predbežne zvoleného smeru otáčania s cieľom aby programátor neboli nútene nastavovať smer otáčania, pokial to nie je dôležité. Ďalšou funkciou tejto metódy je nastavenie externého prerušenia na vstup D3 a príslušného obsluhu prerušenia (angl. Interrupt Service Routine<sup>3</sup>). Režím

<sup>3</sup>abbrev. ISR.



Obr. 1.5: El. schéma zapojenia prístroja MotoShield R1 (prebraté z [11]).

spúšťania externého je nastavený na CHANGE, čo znamená, že ISR bude spustená za každým keď sa na vstupe objavy nábehová alebo dobehová hrana impulzného signálu. Inými slovami, za jednu otáčku zadného hriadeľa ISR bude spustená 14-krát. Obsluha externého prerušenia v sebe zahŕňa jednu inštrukciu, ktorá inkrementuje hodnotu premennej `counter`. Týmto sme vytvorili jednoduché počítadlo impulzov signálu z Hallovho snímača.

Ďalej, v inicializačnej metóde je definovaná aj obsluha cyklického prerušenia. Cyklické prerušenie je nastavené pomocou metód z knižnice `Sampling`, ktorá je súčasťou zdrojového kódu projektu *AutomationShield*. Pomocou metódy `period` (`unsigned long`) si nastavíme periódou cyklického prerušenia a pomocou metódy `interrupt` nastavíme obsluhu cyklického prerušenia. V rámci obsluhy cyklického prerušenia sú spúštané nasledovné inštrukcie.

1. `counted = count`; - načítanie hodnoty počítadla do pamäte
2. `count = 0`; - vynulovanie počítadla
3. `stepEnable = true`; - priradenie kladnej hodnoty pomocnej premennej

Kalibračná metóda `calibration` vykoná krátky experiment. Najprv sa motor roztočí maximálnou rýchlosťou, táto rýchlosť sa za uloží do premennej `maxRPM` pomocou nasledovného príkazu `maxRPM = counted`; . Potom sa motor voľným behom zastaví a zistuje sa minimálny akčný zásah na roztočenie motora zo stavu pokoja pomocou cyklu. V prípade, že sa motor neroztočí pri danom akčnom zásahu,

akčný zásah sa navýsi o jednotku. Konečne, keď sa motor roztočí, načíta sa minimálny akčný zásah do premennej `minDuty` a zároveň aj minimálna uhlová rýchlosť do premennej `minRPM`. Následne sa cyklus preruší a akčný zásah sa vynuluje.

Pri metóde na meranie uhlovej rýchlosťi `sensorReadRPMperc` chceme dostať výstupnú hodnotu v percentách a preto je dôležité poznáť minimálnu uhlovú rýchlosť motora. V tejto metóde sa hodnota počítadla lineárne preškáluje z hodnôt minimálnej a maximálnej uhlovej rýchlosťi na 0 až 100 percent pomocou metódy `mapFloat`, ktorá taktiež pocchádza zo zdrojového kódu projektu *Automationshield*.

Ďalšia metóda na meranie uhlovej rýchlosťi je `sensorReadRPM`, ktorá vráti hodnotu uhlovej rýchlosťi v jednotke  $\text{ot} \cdot \text{min}^{-1}$ .

## 1.3 Zhrnutie nedostatkov

V tejto podkapitole si zhrnieme všetky nedostatky prvej verzie prístroja MotoShield po hardvérovej a softvérovej stránke a v pokračovaní sa ich budeme snažiť odstrániť.

### I Softvér

- Vyriešiť konflikt knižníc zdrojového kódu AutomationShield pri kompliacii knižnice MotoShield.h v systému priebežnej integrácie priebežnej (angl. Continuous Integration<sup>4</sup>)
- Premenovanie metódy `sensorReadRPMperc` na `sensorRead` z dôvodu dodržania pravidel nomenklatúry projektu AutomationShield.
- Zápis metód cyklického a externého prerušenia v podobe lambda výrazu.

### II Hardvér

- Správne zapojenie Hallovho snímača pre prototypizačné dosky Arduino UNO a Mega 2560.
- Návrh jednokanáloveho H-mostíka z dôvodu vylúčenia redundancie.
- Návrh periférie na vylepšenie merania odberu elektrického prúdu motora.
- Návrh el. obvodu s ohľadom na kompatibilitu s doskami architektúry SAM

---

<sup>4</sup>abbrev. CI.

## 2 Návrh vylepšenej verzie hardvéru

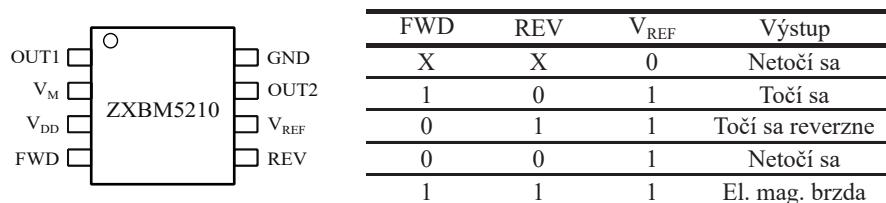
V tejto kapitole bude popísaný kompletný proces návrhu druhej verzie zariadenia MotoShield. Pri návrhu budeme vychádzať z prvej verzie so snahou eliminovať nedostatky uvedené V Kapitole 1 na strane 9.

### 2.1 Návrh elektronických komponentov

#### 2.1.1 Pohon motora

V prvej verzie prístroja MotoShield na pohon motora bol navrhnutý integrovaný obvod L293D. Z hľadiska funkcionality tento komponent spĺňa všetky požiadavky, avšak obsahuje aj druhý kanál, ktorý je v našom uplatnení nevyužitý. Tento komponent je v prevedení PDIP(16), čo znamená, že technológia montáže na dosku plošných spojov (angl. Printed Circuit Board<sup>1</sup>) je Through-Hole Technology<sup>2</sup>. Z dôvodu minimalizácie rozmerov el. obvodov a modernizácie preferujeme elektronické komponenty s technológiou montáže Surface Mount Technology<sup>3</sup>.

Preto v druhej verzie prístroja MotoShield na pohon motora sme navrhli použiť ZXBM5210 od firmy DIODES Incorporated. Jedná sa o výrazne menšiu súčiastku<sup>4</sup> v prevedení SO(8), čo znamená že komponent má osem nožičiek, Obr. (2.1), s technológiou montáže SMT. Vývody OUT1 a OUT2 sú určené na pripojenie spotrebiča, v našom prípade motora. Vývod VM slúži na prívod energie pre spotrebič. Ďalej, vývod VDD je na napájanie vnútornej logiky komponentu. GND je zem, čiže prívod nulovej referenčnej hodnoty. Vývody FWD a REV slúžia na riadenie smeru otáčania a súčasne aj dávkovanie elektrickej energie spotrebiču v podobe PWM signálu.



Obr. 2.1: Integrovaný obvod ZXBM5210 a pravdivostná tabuľka.

<sup>1</sup>abbrev. PCB.

<sup>2</sup>abbrev. THT.

<sup>3</sup>abbrev. SMT.

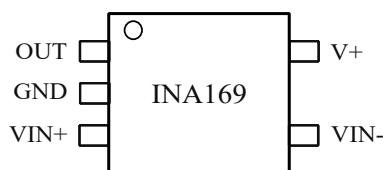
<sup>4</sup>Rozmery komponentu ZXBM5210: (4.95 × 6.10 × 1.75) mm - d×š×v.

V prípade, že by nám bol dostupný digitálno-analógový prevodník (angl. Analog Digital Converter<sup>5</sup>), by sme pomocou vývodu  $V_{REF}$  vedeli na dávkovanie energie využiť vnútorný oscilátor s minimálnou hodnotou 20kHz. V porovnaní s frekvenciou PWM signálu z mikroradiča ATmega382P<sup>6</sup> je frekvencia vnútorného oscilátora viac ako 20-krát vyššia<sup>7</sup>. Pri ovládaní uhlovej rýchlosťi motora pomocou vývodu  $V_{REF}$  nulový činiteľ plnenia (angl. duty cycle) predstavuje napätie 3V a maximálny činiteľ plnenia napätie rovnaké ako je na vývode VDD. Z dôvodu, že používané mikroradičové dosky neobsahujú ADC s rozsahom 3 až 5 V sme túto funkciu obvodu ZXBM5210 nevyužili. Jedna z možností je použiť externý AD prevodník v ďalšej verzii prístroja.

### 2.1.2 Meranie prúdu

Kedže v prvej verzii prístroja bola nevhodne navrhnutá periféria na meranie odberu prúdu, viď Obr. (1.4), sme nútene navrhnúť lepšie riešenie. Pri voľbe boli uprednostnené komponenty s technológiou SMT. Namiesto operačného zosilňovača pre všeobecné účely bol zvolený integrovaný obvod, špecializovaný na meranie odberu prúdu metódou meracieho odporu<sup>8</sup> INA169 od firmy Texas Instruments, v prevedení SOT-23. Jedná sa o zariadenie určené pre tzv. High-Side meranie prúdu, čiže merací odpor je sériovo zapojený pred spotrebičom. Kedže ide o unipolárnu súčasťku, meranie prúdu je možné iba v jednom smere. Toto môže byť problematické, pretože meranie prúdu bude možné iba v kladnom smere otáčania motora. Avšak pre didaktické príklady je takáto zostava postačujúca.

V dátovom hárku obvodu INA169 je uvedené, že na presnejšie meranie výstupného signálu je odporúčané použiť operačný zosilňovač v tzv. „buffer“ zapojení. Pri takomto zapojení do neinvertujúceho vstupu je privedený signál a invertujúci vstup je zapojený na výstupný signál. Takéto zapojenie vynuluje impedanciu analógovo-digitálneho prevodníka na mikroradičoch.



Obr. 2.2: Integrovaný obvod INA169.

---

<sup>5</sup>abbrev. ADC.

<sup>6</sup>980Hz pre výstupy D5 a D6.

<sup>7</sup>Pri pohone jednosmerného motora vyššou frekvenciou PWM dostaneme plynulejší chod motora.

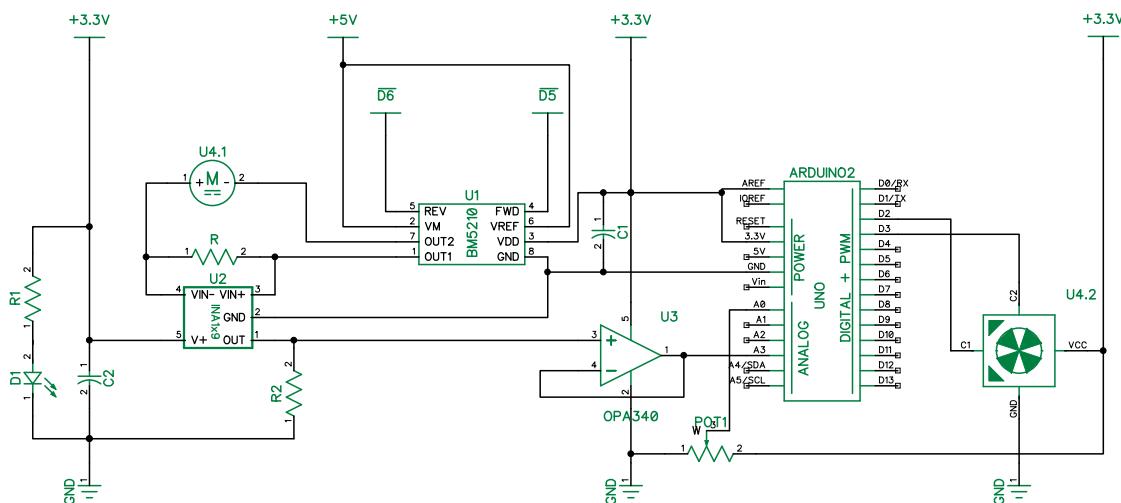
<sup>8</sup>Meranie prúdu pomocou meracieho odporu je vhodné riešenie iba pri obvodoch, resp. spotrebičoch, s nízkym výkonom.

## 2.2 Návrh schémy zapojenia

Pri návrhu hardvéru je nesprávna schéma zapojenia jednou z najčastejších chýb, preto je dôležité pozorne analyzovať dátové hárky, najmä pri použití zložitých integrovaných obvodov. V prípade návrhu komplexnejších elektrických schém je vhodné schému nakresliť v simulačnom prostredí a overiť si správanie sa obvodu. V prípade el. obvodu druhej verzie prístroja MotoShield sme neboli nútení obvod odsimulovať, keďže sa jedná o pomerne jednoduchý obvod.

Jedným z nedostatkov prvej verzie prístroja MotoShield bola nekompatibilita s prototypizačnými doskami, ktoré sú založené na mikroradičoch s CMOS<sup>9</sup> logickou úrovňou 3.3 V. Takéto dosky majú maximálnu toleranciu napäťia v obvode 3.6V a v prípade, že sa vyskytne vyššia hodnota napäťia, dochádza k trvalému fyzickému poškodeniu. Preto pri navrhovaní druhej verzie prístroja budeme navrhovať s dohľadom na to, aby ani jeden vstupný signál nepresiahol hodnotu 3.3 V. Toto zabezpečíme tak, že komponenty s výstupným signálom budeme napájať z vývodu 3.3 V.

Ako bolo už spomenuté, výstupy z Hallovho snímača potrebujú byť pripojené k mikroradiču cez digitálne vstupy, vhodné na externé prerušenie. Pre mikroradiče ATmega382P a ATmega2560 sú na externé prerušenie vyhradené vstupy D2 a D3. Amplitúda výstupného signálu je rovnaká ako hodnota napájania a preto je Hallov snímač zásobovaný zo zdroja 3.3 V, viď Obr. (2.3).



Obr. 2.3: Schéma zapojenia el. obvdou prístroja MotoShield R2.

Integrovaný H-mostík ZXBM5210 môže byť napájaný zo zdroja 5 V, pretože z tohto obvodu nevystupuje žiadny signál. Predsa len, napájanie vnútornej logiky prichádza z 3.3 V zdroja, keďže sa 5 V zdroj používa na roztočenie motora, pričom hodnota napäťia môže klesnúť. Integrované obvody môžu byť citlivé na poruchy napäťia pri napájaní vnútornej logiky a preto je dobrým zvykom pridať kondenzátor, ktorý poruchy vyhľadí. Na takéto účely vystačí kondenzátor s hodnotou  $0.1 \mu\text{F}$ .

<sup>9</sup>abbrev. Complementary Metal–oxide–semiconductor.

Nožičky na ovládanie smeru a rýchlosťi otáčania motora sú na mikroradič pripojené cez výstupy D5 a D6, kde je frekvencia PWM signálu 980 Hz. V prípade, že cez výstup D5 vyšleme signál, motor sa začne otáčať v pravotočivom smere a v opačnom prípade v ľavotočivom, podľa Tab. 2.1. Režim ovládania rýchlosťi otáčania pomocou integrovaného oscilátora bez externého DAC nemôžeme použiť a preto je vývod  $V_{REF}$  pripojený na 5 V, čím dostaneme IC do režimu externého PWM ovládania. V prípade, že by niekto chcel použiť vnútorný oscilátor a pridať externý DAC, prichystali sme verziu PCB, na ktorej je  $V_{REF}$  vyvedený na voľnú kontaktovú podložku s dierkou.

Ked' sa pozrieme na schému zapojenia, Obr. (2.3), uvidíme, že merací odpor  $R$  je do série zapojený s motorom. Pod meracím odporom sa nachádza IC na meranie prúdu INA169, ktorého vývody  $VIN+$  a  $VIN-$  sú napojené pred a za merací odpor. IC tieto signály odčíta. Odčítaný signál je vysielaný cez vývod OUT a zároveň aj zosilnený v závislosti od hodnoty odporu  $R_2$ , podľa vzťahu v Rov. (2.1), kde  $R_S$  je hodnota meracieho odporu a  $R_L$  je hodnota odporu na určenie zosilnenia v jednotke  $\Omega$ .  $V_{OUT}$  je hodnota výstupného signálu v jednotke V a  $I_S$  je hodnota el. prúdu v jednotke A. Symbolom  $g_m$  je označená transkonduktancia<sup>10</sup> integrovaného obvodu. Hodnota transkonduktancie pre tento snímač je  $10^{-3} \text{ S}$  [20].

$$V_{OUT}(t) = I_S(t) \cdot R_S \cdot R_L \cdot g_m. \quad (2.1)$$

Po explicitnom vyjadrení veličiny prúdu z Rov. (2.1) a následným dosadením hodnôt odporov a transkonduktancie, dostaneme vzťah na prevod z výstupného napäťa na odber prúdu motora v základných jednotkách, viď Rov. (2.2).

$$I_S(t) = \frac{V_{OUT}(t)}{R_S \cdot R_L \cdot g_m} = \frac{V_{OUT}(t)}{10\Omega \cdot 10^4\Omega \cdot 10^{-3}\Omega^{-1}} = \frac{V_{OUT}(t)}{100\Omega} \quad (2.2)$$

Ako bolo už spomenuté, podľa pokynu z dátového hárku sme zahrnuli aj operačný zosilňovač OPA340 v prevedení SOT-23 v „buffer“ zapojení na elimináciu impedancie ADC. Konečne, výstupný signál je pripojený mikroradiču cez analógový vstup A3.

V schéme je zakreslený aj potenciometer POT1 s hodnotou  $10 \text{ k}\Omega$ . Napájaný je zo zdroja 3.3 V a bežec je k mikroradiču pripojený cez vstup A0.

Pri doskách Arduino UNO a Mega 2560 je prednastavená referenčná hodnota AD prevodníka 5 V a keďže naše vstupné signály majú najvyššiu hodnotu 3.3 V, referenčnú hodnotu na AD prevodníku nastavíme na 3.3 V pripojeným tejto hodnoty na vstup AREF.

Na Obr. 2.3 si na schéme môžeme všimnúť, že na vyhľadenie napájacieho signálu boli pri IC ZXBM5210 a INA169 boli použité nepolarizované kondenzátory C1 a C2. Hodnota kapacity obidvoch kondenzátorov je rovnaká,  $0.1 \mu\text{F}$ , uvedená v dátových hárkoch [20, 8].

Taktiež si si všimneme v schéme zapojenia, že bola použitá aj svetelná dióda (angl. Light-emmiting Diode<sup>11</sup>) D1, sériovo zapojená za príslušným odporom  $1 \text{ k}\Omega$ .

<sup>10</sup>Transkonduktancia je pomer prúdu na výstupe vzhľadom na zmenu, resp. rozdiel napäťa na vstupe.

<sup>11</sup>abbrev. LED.

## 2.3 Návrh dosky plošných spojov

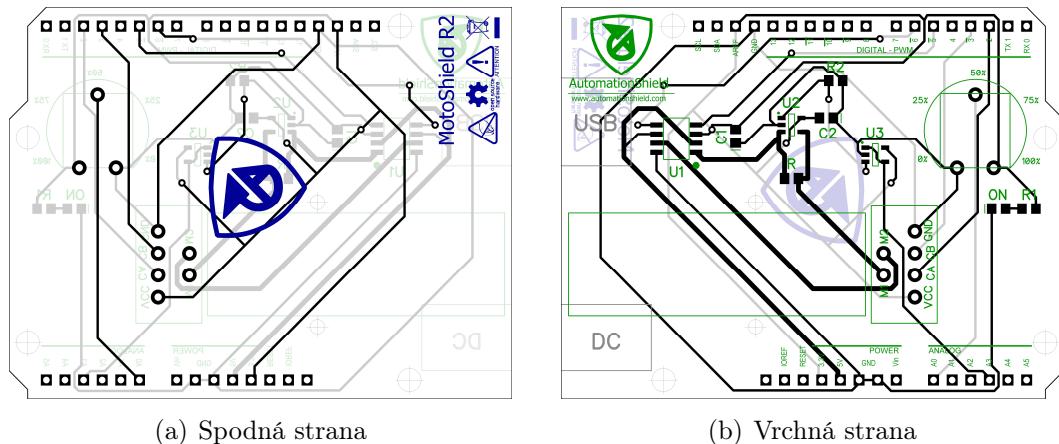
Dosku plošných spojov sme navrhovali, rovnako ako aj schému el. obvodu, v programe DipTrace. Na kreslenie el. schémy sa v programe DipTrace používa prostredie *Schematics*. Toto prostredie pomocou príkazu *Convert to PCB* umožňuje automatizované vytvoriť dosku plošných spojov na základe schémy zapojenia. Pred spustením príkazu je dôležité skontrolovať, či každej súčaistke prislúcha zodpovedajúci odtlačok (angl. Footprint) na ktorý naspájkujeme komponent.

Pre každý komponent sú minimálne a maximálne rozmery odtlačku uvedené v dátovom hárku. Pri ručnom spájkovaní je lepšie si zvoliť maximálne rozmery odtlačku. Toto výrazne obľahčí proces spájkovania.

Všetky komponenty s dvomi kontaktmi sme zvolili v prevedení 0805 a preto sú všetky odtlačky pre tieto komponenty rovnaké. Pre IC ZXBM5210 je v prevedení SO(8) a INA169 spolu s OPA340 sú v prevedení SOT-23. Keďže sú integrované obvody pomerne malých rozmerov na ručné spájkovanie, šírku podložiek sme si zvolili o 20 percent väčšiu. Káblíky z motorovej jednotky sú pretiahnuté cez podložky s dierkou a zaliate spájkou. Potenciometer je jediný elektrický komponent montovaný s technológiou montáže THT.

Cesty na PCB, cez ktoré prúdi väčšie množstvo el. energie, sú širšie z dôvodu zníženia vnútorného odporu. Pri navrhnutom el. obvode sa toto prejavuje iba medzi IC na pohon motora a samotným motorom. Šírka týchto ciest je dvakrát väčšia od ostatných, pričom širšie cesty majú rozmer 0.66 mm a ostatné 0.33 mm. Pri návrhu ciest, cez ktoré prúdi signál s vyššou frekvenciou, je odporúčané sa vyhnúť pravým uhlom z dôvodu výskytu elektromagnetickej indukcie [2].

Hrúbka navrhnutého PCB je 1.6 mm, rovnaká ako pri mikroradičovej doske Arduino. Rozmery PCB sú taktiež rovnaké ako pri doske Arduino. Obrys PCB, spolu s kontaktnými podložkami, pomocou ktorých nasadíme prístroj na mikroradičovú dosku, sme prevzali z knižnice komponentov *AutomationShield* pre prostredie DipTrace.



Obr. 2.4: Doska plošných spojov - MotoShield R2.

## 2.4 Cena

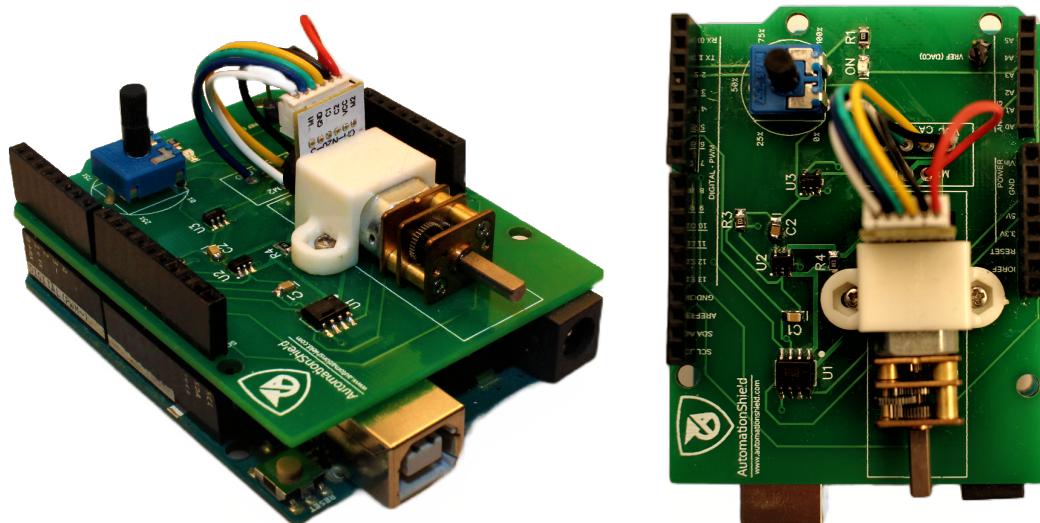
Pomerne nízka cena didaktických prístrojov projektu *AutomationShield* je pre verejnosť jedna z najatraktívnejších vlastností. V Tab. 2.1 sú uvedené ceny a zhrnutie ceny všetkých komponentov druhej verzie prístroja MotoShield aj vrátane dosky plošných spojov.

Tabuľka 2.1: Zoznam a cena komponentov v Eurách<sup>a,b</sup>.

Názov	Popis	Označenie	Ks.	Cena	Spolu
Motor	DFRobot Micro motor s prevodokou a enkóderom - 6V 41RPM 380:1	U4	1	11.31	11.31
Pohon motora	DIODES Incorporated ZXBM5210 pohon motora na riadenie rýchlosťi	U1	1	0.75	0.75
Senzor prúdu	Texas Instruments INA169 - SOT23	U2	1	3.14	3.14
Operačný zosilňovač	Texas Instruments OPA340 - univerzálny operačný zosilňovač - SOT-23	U3	1	3.04	3.04
Potenciometer	10 kΩ, 150 mW (CA9MVV 10K)	POT1	1	0.11	0.11
Rezistor	0805, 10 Ω, 0.5 W, 1%	R	1	0.62	0.62
Rezistor	0805, 1k Ω	R1	1	0.15	0.15
Rezistor	0805, 10k Ω, 1%	R2	1	0.15	0.15
Kondenzátor	0805, keramický, 100 nF	C1, C2	2	0.01	0.02
LED	Dióda, zelená farba	D1	1	0.18	0.18
PCB	2 vrstvy, FR4, hrúbka 1.6 mm	-	1	0.45	0.45
					<b>Spolu: 19.92€</b>

<sup>a</sup> Uvedená je cena pre objednávky v máлом množstve.

<sup>b</sup> V cene nie je zarátané poštovné.



Obr. 2.5: MotoShield R2 pripojený na Arduino UNO.

# 3 Vývoj programátorského rozhrania pre MotoShield R2

Pri návrhu programátorského rozhrania pre druhú verziu prístroja MotoShield sme vychádzali z programátorského rozhrania prvej verzie prístroja, ktoré bolo napísané v rámci bakalárskej práce *MotoShield: Vývoj programátorského rozhrania, identifikácia, riadenie a úprava didaktického prístroja na riadenie rýchlosťi motorov*[4]. V práci bolo navrhnuté programátorské rozhranie pre prostredie Arudino IDE v jazyku C++ a taktiež aj pre prostredia MATLAB a Simulink.

API pre druhú verziu pridáme k existujúcemu rozhraniu tak, aby si programátor vedel jednoduchým spôsobom zvoliť, ktorú verziu prístroja používa. Taktiež bude v tejto kapitole popísaný aj spôsob vyriešenia nedostatkov uvedených v kapitole 1 na strane 9.

## 3.1 Riešenie konfliktov API pre MotoShield R1

Programátorske rozhranie prvej verzie prístroja pre prostredie Arduino IDE obsahovalo konflikt pri pripojení k zdrojovému kódu projektu AutomationShield. Konflikt bol indikovaný softvérom kontinuálnej integrácie Travis CI v rámci nástroja na distribuované verziovanie softvéru GitHub. Pri snahe spojenia vetiev kódu sa vykonáva tzv. „Smoke Test“, čo znamená vykonanie predbežných testov funkčnosti medzi ktorými je aj komplilácia, pri ktorej konflikt nastal. Kompilátor indikoval, že premenné `count`, `counted` a `stepEnable` sú deklarované ako `inline`. Toto bolo použité s cieľom deklarácie a zároveň aj inicializácie premennej v rámci triedy, čím dostaneme koncínnejší a prehľadnejší kód. Pomôcka Travis CI funguje na takom princípe, že sa vytvorí virtuálny počítač s operačným systémom, v tomto prípade ubuntu, nainštaluje sa Arduino IDE a následne skúšame kompilovať súbory. Bohužiaľ, prostredie Arduino IDE pre operačné systém linux používa verziu jazyka C++11 a identifikátor `inline` je podporovaný až od verzie C++17 [6]. Preto uvedené premenné museli byť inicializované mimo triedy `MotoShieldClass` nasledovne:

```
volatile uint16_t MotoShieldClass::count; //--initializing static variables  
volatile uint16_t MotoShieldClass::counted;  
volatile bool MotoShieldClass::stepEnable;
```

Ďalším konfliktom knižnice `MotoShield` bolo obsadenie rovnakého časovača ako knižnica `Sampling`. Toto nastalo z dôvodu, že Arduino IDE, pokial nájde hlavičkový súbor „.h“ s príslušným zdrojovým súborom „.cpp“, ich prekompileluje súbory

bez ohľadu na to, či sa niekde používajú. Tento konflikt sme vyriešili tým, že sme obsah zdrojového súboru vložili do hlavičkového. Preto programátorské rozhranie pre prístroj MotoShield bude obsahovať iba hlavičkový súbor „.h“.

## 3.2 API pre prostredie Arduino IDE

Po vyriešení konfliktov API môžeme pokračovať v písaní programátorského rozhrania pre druhú verziu prístroja MotoShield.

Kedže sa programátorské rozhranie pre prvú a druhú verziu prístroja budú mierne odlišovať, je potrebné kód rozdeliť. V jazyku c++ vieme kód elegantným spôsobom rozdeliť za účelom toho, aby si programátor mohol zvoliť akú verziu prístroja používa. Optimálnym riešením je použiť tzv. preprocesorovú direktívou podmienenia. Preprocesorové direktívy sa vykonávajú pred samotnou kompliaciou, a preto sa kód mimo splnenej podmienky ani len nebude komplikovaný. Výraz preprocesorovej podmienky je nasledovný:

```
#if SHIELDRELEASE == 2
    vykonajPrikaz();
#endif
```

Volaná funkcia `vykonajPrikaz` bude spustená iba v prípade, že symbolická konštanta `SHIELDRELEASE` nadobúda hodnotu 2. Symbolické konstanty sa môžu definovať nasledovne:

```
#ifndef SHIELDRELEASE
#define SHIELDRELEASE 2
#endif
```

Týmto kódom sme zabezpečili, aby symbolická konštanta nadobudla hodnotu 2 iba v prípade, že už niekde jej hodnota nebola definovaná. Takto umožníme programátorovi, aby si mohol zadefinovať verziu používanejho prístroja v súbore, z ktorého volá API. Pokiaľ si programátor nezadefinuje verziu používanejho prístroja, verzia bude považovaná za najnovšiu.

Preprocesorová direktíva podmienenia je použitá aj na vyhýbanie sa chybám, ktoré by vznikli pri viačnásobnom zahrnutí programátorského rozhrania do programu, tzv. „Include Guard“. V tele podmienky je zahrnutý celý kód programátorského rozhrania. Podmienka vyzerá nasledovne:

```
#ifndef MOTOSHIELD_H
#define MOTOSHIELD_H
// programátorské rozhranie
#endif
```

V knižniciach a literatúre sa, za týmto účelom, objavuje aj použitie direktívi `pragma`, ktorá je podporovaná pri novších komplikátoroch. V prípade klasického „Include Guardu“ je symbolickú konštantu možné použiť iba pre jeden hlavičkový súbor, kým pri direktíve `pragma` sa konštanta môže opakovať.

```
#pragma once
// programatorske rozhranie
```

Pri písaní rozhrania pre prístroj je nutné dodržiavať pravidlá písania programátorských rozhraní projektu **AutomationShield**. Najmä, kompletne programátorské rozhranie by malo byť zahrnuté do jednej triedy s menom rovnakým ako je meno prístoroja a príponou „Class“. Trieda je v zásade iba programom definovaný dátový typ a ich „premenné“, resp. inštancie sa nazývajú objekty. Objekty sú schopné disponovať iba elementami s verejným (angl. public) prístupom. Po kiaľ nedefinujeme verejný prístup metód a premenných, prístup bude považovaný za súkromný (angl. private) a objekt nebude schopný volať jednotlivé metódy. Pri paradigme objektovo orientovaného programovania je najlepším zvykom deklarovať ako verejné iba elementy, ktoré nemôžu byť súkromné z dôvodu zamedzenia porúch prvkov podstatných pre rozhranie.

```
class MotoShieldClass{ // definicia triedy
    public:
        // verejne entity triedy
    private:
        // súkromne entity triedy
};
```

Uvedená trieda obsahuje iba jeden objekt pomenovaný po prístroji. Objekt je vytvorený v rámci API, čo znamená, že po zahrnutí API do programu, objekt bude už vopred vytvorený.

```
MotoShieldClass MotoShield; // vytvorenie objektu MotoShield
```

### 3.2.1 Konštruktor

Konštruktor je metóda pomenovaná rovnako ako trieda, ktorá sa spustí ihneď po vytvorení objektu z danej triedy [12]. Pri druhej verzie prístroja sme na vstup AREF priviedli referenčnú hodnotu napäťia pre analógovo digitálny prevodník. Na to, aby mikrokontróler bral referenciu na základe napäťia na svorke AREF je potrebné vykonať príkaz **analogReference** so vstupným argumentom **EXTERNAL**.

```
MotoShieldClass(){ // Constructor
    analogReference(EXTERNAL);
}
```

Funkcia **analogReference** je volaná v konštruktore z nasledovného dôvodu:

*V prípade, že na svorku AREF je privodené napätie, čiže ak by prístroj bol vsadený do prototypizačnej dosky a zároveň by programátor zavolal najprv funkciu **analogRead**, napätie na svorke by sa sčítalo s vnútorné generovaným napäťím referencie AD prevodníka. Toto by mohlo zapríčiniť trvalé poškodenie mikroradiča! [1].*

Volaním funkcie **analogReference** v rámci konštruktora zostáva hrozba bezpečenstva iba ak by študent meral napätie na ADC bez toho, aby zahrnul API do programu.

### 3.2.2 Metódy na ovládanie motora

Na ovládanie motora pre MotoShield R2 boli napísané nasledovné metódy:

- `setDirection` - nastavenie smeru otáčania,
- `actuatorWrite` - vysielanie akčného zásahu v percentách,
- `actuatorWriteVolt` - vysielanie akčného zásahu vo voltoch,

#### Nastavenie smeru otáčania

Druhá verzia prístroja MotoShield na pohon motora používa ZXBM5210. Smer otáčania pri tomto IC je určený podľa toho, na ktorú svorku vysielame PWM signál. Preto, v rámci metódy `setDirection` bude nastavovaná hodnota premennej `_direction`. Funkcia `setDirection` čaká jeden vstupný argument dátového typu `bool`, pričom kladná hodnota symbolizuje pravotočivý smer otáčania a napäk. V rámci metód na vysielanie akčného zásahu sa budeme odvolávať na hodnotu premennej `_direction` podľa ktorej určíme číslo výstupu, cez ktorý sa bude vysieláť akčný zásah.

```
void MotoShieldClass::setDirection(bool direction = true){  
    MotoShieldClass::_direction=direction;  
}
```

#### Vysielanie akčného zásahu

Na definovanie akčného zásahu boli navrhnuté dve metódy. Prvá, `actuatorWrite` požaduje jeden vstupný parameter. Tento parameter predstavuje hodnotu akčného zásahu v percentách. Na vylúčenie saturácie motora<sup>1</sup>, rovnako ako pri API pre prvú verziu prístroja, je použitá podmienka na minimálny akčný zásah. Hodnota minimálneho akčného zásahu je zapísaná do premennej `minVolt`, ktorej hodnotu zistujeme v kalibračnej metóde. Ďalšia podmienka zistuje hodnotu smeru otáčania porovnaním premennej `_direction`.

```
void MotoShieldClass::actuatorWrite(float percentValue) {  
    if(percentValue < minDuty && percentValue != 0)  
        percentValue = minDuty; // minimalny akcny zasah  
    if(MotoShieldClass::_direction){ // zistenie smeru otacania  
        analogWrite(MOTO_UPIN1, AutomationShield.percToPwm(percentValue));  
        analogWrite(MOTO_UPIN2, 0);  
    }else{  
        analogWrite(MOTO_UPIN1, 0);  
        analogWrite(MOTO_UPIN2, AutomationShield.percToPwm(percentValue));  
    }  
}
```

Príkazom `analogWrite` vysielame akčný zásah, kde prvý argument je číslo výstupu. `MOTO_UPIN1` a `MOTO_UPIN2` sú symbolické konštenty definované preprocessorovou direktívou, ktorých hodnoty sú D5 a D6. Druhý argument je hodnota

---

<sup>1</sup>Na roztočenie motora potrebujeme vyslať minimálny akčný zásah.

činiteľa plnenia PWM signálu v 8-bitovom rozlíšení. Metóda `percToPwm` slúži na prevod z percent na 8-bitovú stupnicu. Metóda `percToPwm` je prebratá zo zdrojového kódu projektu `AutomationShield`.

Druhá metóda na zápis akčného zásahu, `actuatorWriteVolt`, sa od predchádzajúcej metódy líši jedine prevodom hodnoty vstupného argumentu na 8-bitové číslo, podľa vzťahu v Rov. (1.1) na strane 7.

```
void MotoShieldClass::actuatorWriteVolt(float voltageValue){
    if(voltageValue < minVolt && voltageValue != 0) voltageValue = minVolt;
    if(MotoShieldClass::_direction){
        analogWrite(MOTO_UPIN1,sq(voltageValue)*255.0/sq(REF));
        analogWrite(MOTO_UPIN2,0);
    }else{
        analogWrite(MOTO_UPIN1,0);
        analogWrite(MOTO_UPIN2,sq(voltageValue)*255.0/sq(REF));
    }
}
```

### 3.2.3 Metódy na meranie výstupných veličín

Výstupné veličiny systému sú uhlová rýchlosť motora a odber el. prúdu.

#### Meranie uhlovej rýchlosťi

Na meranie uhlovej rýchlosťi sú v rámci API nasledovné metódy:

1. `sensorRead` - meranie uhlovej rýchlosťi v percentách
2. `sensorReadRMP` - meranie uhlovej rýchlosťi v otáčkach za minútu
3. `sensorReadRadian` - meranie uhlovej rýchlosťi v radiánoch za sekundu

Metóda `sensorRead` je iba premenovaná metóda `sensorReadRMPperc`, ktorá bola napísaná v rámci programátorského rozhrania pre prvú verziu prístroja. Metóda je bližšie popísaná v bakalárskej práce *MotoShield: Vývoj programátorského rozhrania, identifikácia, riadenie a úprava didaktického prístroja na riadenie rýchlosťi motorov*[4]. Taktiež, aj metóda `sensorReadRPM` je v úplnosti prebratá z rovnakej práce.

Metóda `sensorReadRadian` vráti hodnotu uhlovej rýchlosťi v jednotke radián za sekundu. Základnú informáciu o uhlovej rýchlosťi nesie premenná `counted`. Hodnota tejto premennej predstavuje hodnotu počítadla obsluhy externého prerušenia. Pomocou (`float`) dočasne zmeníme typ premennej `counted` na `float`.

```
float MotoShieldClass::sensorReadRadian(){
    return (float)counted/TICKS*_K/60.0*2.0*PI;
}
```

Vzťah, pomocou ktorého vypočítame hodnotu v jednotke radián za sekundu je obiahnutý v Rov. (3.1), kde  $\omega(t)$  je uhlová rýchlosť v radiánoch za sekundu,  $C_{Ts}(t)$

je počet impulzov za jednu vzorkovaciu periódou. Ďalej,  $T_s$  je vzorkovacia perióda a  $P_{\text{pr}}$  je počet impulzov na jednu otáčku :

$$\omega(t) = \frac{C_{Ts}(t)}{T_s \cdot P_{\text{pr}}} \cdot 2\pi \quad (3.1)$$

Konšanta  $P_{\text{pr}}$  je v programe predstavená symbolickou konštantou **TICKS**, ktorej hodnota sa mení na základe verzie prístroja. Keďže magnetické koliesko má sedem párov pólov, za jednu otáčku dostaneme sedem impulzov na jednom kanále Hallovho snímača. V inicializačnej metóde je nastavený režim spúšťania externého prerušenia **CHANGE**, čiže detegujeme aj nábehové, aj dobehové hrany impulzného signálu. Inak povedané, pri jednej otáčke hriadeľa sa 28-krát spustí obsluha externého prerušenia, čiže hodnota  $P_{\text{pr}}$  je 28. Pri prvej verzii prístroja je aktívny iba jeden kanál a preto sa obsluha prerušenia (angl. Interrupt Service Routine<sup>2</sup>) spustí iba 14-krát, čiže hodnota  $P_{\text{pr}}$  je 14.

### Meranie odberu el. prúdu

V druhej verzii prístroja je na meranie prúdu použitý snímač INA169 a príslušná periféria. Odber prúdu je interpretovaný napäťovým signálom podľa vzťahu v Rov (2.2), privedeným k mikroradiču cez analógovo digitálny prevodník.

Na meranie prúdu bola napísaná metóda **sensorReadCurrent**, ktorá vrati hodnotu odberu prúdu v základnej jednotke SI sústavy. Hodnotu napäťia z AD prevodníka vyčítame pomocou príkazu **analogRead**, kde prvý argument je číslo analógového vstupu. Pri druhej verzii prístroja je prúd meraný na vstupe A3. Táto hodnota je v API zapísaná do symbolickej konštanty **MOTO\_YPIN**. Preškálovanie z číselnej hodnoty AD prevodníka na napätie je uskutočnené delením s hodnotou rozlíšenia AD prevodníka a vynásobením referenčnou hodnotou na AD prevodníku. Rozlíšenie AD prevodníka je rôzne v závislosti od použitej prototypizačnej dosky. Hodnota rozlíšenia ADC je predstavená symbolickou konštantou **ADCREF**, ktorej hodnota je nastavená podľa architektúry mikroradiča. Referenčná hodnota ADC je 3.3 V.

```
float MotoShieldClass::sensorReadCurrent() {
    return analogRead(MOTO_YPIN)/ADCREF*AREF3V3/100.0;
}
```

### 3.2.4 Pomocné metódy

#### Inicializačná metóda

Pri použití didaktických prístrojov projektu *AutomationShield* v prostredí Arduino IDE by programátor mal najprv zavolať inicializačnú metódu **begin**. V rámci tejto metódy sú nastavené všetky parametre, ktoré nám zaručia správnu funkcionality ostatných metód programátorského rozhrania.

---

<sup>2</sup>abbrev. ISR.

Najprv, je v rámci metódy `begin` nastavený režim každého použitého vstupu a výstupu. Nastavenie vstupov a výstupov nám umožňuje príkaz z natívnej knižnice Arduino, `pinMode`. Tento príkaz vyžaduje dva vstupné argumenty, z ktorých prvý predstavuje číslo nastavovaného „pin-u“ napr. D5, D10, A0, atď. Druhým argumentom je režim, či daný „pin“ chceme nastaviť ako vstupný alebo výstupný, čo je symbolizované slovami `INPUT` a `OUTPUT`. Ďalej je v inicializačnej metóde zavolaná metóda `setDirection` s kladným argumentom s cieľom zvoliť prednastavený pravotočivý smer otáčania v prípade, že programátoru nezáleží na určeniu smeru.

Jediný vstupný argument metódy `begin` je číslo, ktoré predstavuje hodnotu vzorkovacej periody pre cyklické prerušenie. Hodnotu cyklického prerušenia nastavujeme pomocou metódy `period`, volanej cez objekt `Sampling`, ktorý pochádza z rozhrania Sampling knižnice `AutomationShield`. Metóda `period` vyžaduje uvedenie vzorkovacej periody v jednotke s, kým do vstupného argumentu je očakávaná hodnota vzorkovacej periody v ms. Preto je argument vynásobený číslom  $10^3$ . V prípade, že užívateľ nezadá vstupný argument pri volaní inicializačnej metódy, vzorkovacia perióda bude mať prednastavenú hodnotu 20 ms.

Nastavenie obsluhy cyklického prerušenia nám umožňuje metóda `interrupt`, triedy `Sampling`. Táto za vstupný parameter očakáva uvedenie tzv. „pointer“ funkcie, resp. metódy. V API pre prvú verziu prístroja bola napísaná metóda obsluhy cyklického prerušenia `_InterruptSample`, obsahom ktorej boli inštrukcie uvedené na strane 8. Namiesto zbytočného vytvorenia tejto metódy sme použili tzv. „lambda“ výraz [10]. Sú to v zásade nemenované funkcie, ktoré vieme zapísať do jedného riadku pre vstupný argument ako má metóda `interrupt`. Zápis jednotlivých inštrukcií v podobe „lambda“ výrazu vyzerá nasledovne:

```
Sampling.interrupt([]{counted=count; count=0; stepEnable=true;});
```

Poslednou záležitosťou inicializačnej metódy je nastavenie externého prerušenia. Externé prerušenie v rámci knižnice Arduino sa nastavuje pomocou príkazu `attachInterrupt`, kde ako prvý argument uvedieme číslo „pin-u“ na ktorom sa bude detegovať impulzný signál. Druhý argument je obsluha cyklického prerušenia, ktorú sme v našom prípade zapísali „lambda“ výrazom. Posledný argument je na definovanie režimu volania obsluhy externého prerušenia, pričom sme si zvolili prvý režim z uvedených:

- **CHANGE** - ISR sa spustí pri nábehovej a dobehovej hrany impulzu
- **RISING** - ISR sa spustí pri nábehovej hrany impulzu
- **FALLING** - ISR sa spustí pri dobehovej hrany impulzu
- **HIGH** - ISR sa spustí pri logickej jednotke
- **LOW** - ISR sa spustí pri logickej nule

Dôvodom zvolenia režimu **CHANGE** je ten, že dostaneme najviac impulzov na otáčku hriadeľa, čo nám umožní použiť menšiu vzorkovaciu periódu.

Kedže máme Hallov inkrementálny snímač s dvomi kanálmi, potrebujeme 2-krát nastaviť externé prerušenie, pre obidva kanály. Obsluha externého prerušenia je

rovnaká pre obidva „pin-y“, pričom vlastne prírastkovo navyšujeme (angl. Increment) hodnotu premennej `count` o hodnotu 1.

Inicializačná metóda je definovaná nasledovne:

```
void MotoShieldClass::begin(float _Ts = 20.0){
    pinMode(MOTO_YPIN, INPUT); // nastavenie rezimu pinov
    pinMode(MOTO_UPIN1, OUTPUT);
    pinMode(MOTO_UPIN2, OUTPUT);
    pinMode(MOTO_RPIN, INPUT);
    pinMode(MOTO_YPIN1, INPUT);
    pinMode(MOTO_YPIN2, INPUT);
    setDirection(true); // prednastaveny smer otacania
    Sampling.period(_Ts*1000.0); // perioda cyklickeho prerusenia
    _K=60000.0/_Ts;
    // obsluha cyklickeho prerusenia
    Sampling.interrupt([]{counted = count;count = 0;stepEnable = true;});
    // obsluha externeho prerusenia
    attachInterrupt(digitalPinToInterrupt(MOTO_YPIN1), []{count++;}, CHANGE);
    attachInterrupt(digitalPinToInterrupt(MOTO_YPIN2), []{count++;}, CHANGE);
}
```

## Kalibračná metóda

Pod kalibráciou pre prístroj MotoShield chápeme zisťovanie hodnoty uhlovej rýchlosťi pri maximálnom a minimálnom akčnom zásahu. Preto, kalibračná metóda vykoná sekvenciu, resp. krátky experiment, v ktorom sa motor roztočí maximálnou uhlovou rýchlosťou, ktorej hodnota sa zapíše do pomocnej premennej `maxRPM`. Následne sa motor zastaví a v rámci cyklu sa snažíme motor roztočiť s minimálnym akčným zásahom. V prípade, že sa motor začne otáčať, cyklus sa zastaví, minimálna uhlová rýchlosť sa načíta do premennej `minRPM` a načíta sa aj hodnota minimálneho akčného zásahu do premennej `minDuty`. Kalibračná metóda bola napísaná v API pre prvú verziu prístroja. Rovnaká metóda je použitá aj pre druhú verziu prístroja. Definíciu metódy a bližší popis najdete v bakalárskej práci [4].

## Čítanie referencie

Na čítanie polohy potenciometra, resp. referenčnej hodnoty bola napísaná metóda `referenceRead`.

```
float MotoShieldClass::referenceRead() {
    return AutomationShield.mapFloat( (float)analogRead(MOTO_RPIN), 0.0, ADCREF,
    ↳ 0.0, 100.0);
}
```

Táto metóda vráti hodnotu natočenia potenciometra v percentách. Hodnota z ADC je vycítaná pomocou príkazu `analogRead` a následne je preškálovaná pomocou metódy `mapFloat`. Na to, aby sme zachovali kompatibilitu prístroja, resp. programátorského rozhrania s doskami rôznej architektúry, vratnú hodnotu preškálujeme z rozsahu 0 až `ADCREF`<sup>3</sup> na hodnotu v rozsahu 0 až 100 percent.

---

<sup>3</sup>Rozlíšenie analógovo-digitálneho prevodníka - pre mikroradiče architektúry AVR 10-bit, SAM a SAMD 12-bit.

### 3.3 API pre prostredie MATLAB

V prostredí MATLAB sme schopný programovať mikroradičové dosky rodiny Arduino pomocou balíku *MATLAB Support Package for Arduino Hardware*. Programátorské rozhranie pre prvú verziu prístroja prístroja bolo napísané v bakalárskej práci [4]. V tejto kapitole sa budeme snažiť doplniť programátorské rozhranie pre druhú verziu prístroja, tak, aby zostala možnosť voľby verzie prístroja.

Celé programátorské rozhranie je obsiahnuté v rámci triedy pomenovanej podľa prístroja, `MotoShield`. V tele príkazu `properties(Acces = public)` sú deklarované verejne dostupné vlastnosti triedy. Sem sme deklarovali objekt `arduino`, vyhradený na inicializáciu mokroradičovej dosky a objekt `encoder` na spojazdnenie inkrementálneho snímača prostredníctvom knižnice `rotaryEncoder`. Ďalej sme zadefinovali pomocné premenné `minDuty`, `minRPM`, `maxRPM` a `direction`. Funkcionalita týchto premenných je rovnaká ako pri API pre vývojové prostredie Arduino IDE. Taktiež je deklarovaná aj premenná `SHIELDRELEASE`, pomocou ktorej budeme rozoznávať verziu použitého prístroja MotoShield. Konečne, medzi vlastnosti sme deklarovali aj premenné, ktoré predstavujú čísla jednotlivých „pin-ov“. V API pre Arduino IDE sme tieto čísla označovali pomocou symbolických konštánt na úrovni preprocesora.

Ďalšia skupina vlastností triedy sú konštanty, deklarované v rámci `properties(Constant)`. Tu sme definovali hodnotu `PPR`, ktorá predstavuje počet párov pólov inkrementálneho snímača.

Všetky metódy triedy sú definované pomocou príkazu `methods(Access = public)`, čo znamená, že metódy sú verejné a dostupné objektu triedy. Prvá definovaná metóda je tzv. konštruktor. Tento má rovnakú úlohu aj v prípade jazyku C++. Definovaný konštruktor očakáva vstupný argument `release`. Tento argument symbolizuje verziu prístroja, pričom očakávané hodnoty sú 'R1' alebo 'R2'. V prípade, že užívateľ nezadá verziu prístroja pri tvorbe objektu, zoberie sa predvolená hodnota najnovšej verzie prístroja 'R2'. V tele metódy sa potom v závislosti od verzie prístroja definuje hodnota premenných `MOTO_RPIN` a `VREF`, pretože sa tieto vlastnosti odlišujú pri prvej a druhej verzii prístroja.

```
function MotoShieldObject = MotoShield(release)
if nargin < 1
MotoShieldObject.SHIELDRELEASE='R2';
else
MotoShieldObject.SHIELDRELEASE=release;
end
if ~strcmp(MotoShieldObject.SHIELDRELEASE,'R1') ||
~ strcmp(MotoShieldObject.SHIELDRELEASE,'R2'))
error("Available versions of MotoShield are 'R1' and 'R2'")
end
switch MotoShieldObject.SHIELDRELEASE
case 'R1'
MotoShieldObject.MOTO_RPIN = 'A4';
MotoShieldObject.VREF=5;
case 'R2'
MotoShieldObject.MOTO_RPIN = 'A0';
MotoShieldObject.VREF=3.3;
```

```
end  
end
```

Druhá v poradí je inicializačná metóda `begin`. V rámci inicializačnej metódy vytvárame, už spomenuté, objekty pre mikroradičovú dosku a pre inkrementálny snímač. Objekt pre mikroradičovú dosku je vytvorený pomocou príkazu `arduino`. Pri vytvorení tohto objektu sme nútene vo forme vstupného argumentu špecifikovať, akú dosku používame a číslo sériového portu, cez ktorý je doska pripojená k počítaču. Tieto informácie užívateľ uvedie vo forme vstupných argumentov pri volaní inicializačnej metódy `begin`. V rámci spomenutého príkazu je možnosť aj zahrnúť externé knižnice a keďže používame inkrementálny snímač, zahrnieme knižnicu `rotaryEncoder`. Pri druhej verzii prístroja používame externú referenciu napäťia pre ADC. Preto sme pri vytvorení objektu nastavili vlastnosť `AnalogReferenceMode` v režime `external`.

V rámci inicializačnej metódy je vykonané aj konfigurovanie „pin-ov“ mikroradiča, pomocou príkazu `configurePin`. Taktiež je volaná metóda `setDirection`, na definovanie predvoleného smeru otáčania motora. Z dôvodu lepšej prehľadnosti bude uvedená iba časť kódu pre druhú verziu prístroja.

```
function begin(MotoShieldObject, Port, Board)  
MotoShieldObject.arduino = arduino(Port,Board,'Libraries', 'rotaryEncoder',  
    ↳ 'AnalogReferenceMode', 'external', 'AnalogReference', 3.3);  
configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_UPIN1,'PWM');  
configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_UPIN2,'PWM');  
configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_YPIN,'AnalogInput');  
configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_YPIN1,'Interrupt');  
configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_YPIN2,'Interrupt');  
configurePin(MotoShieldObject.arduino,MotoShieldObject.MOTO_RPIN,'AnalogInput');  
MotoShieldObject.encoder = rotaryEncoder(MotoShieldObject.arduino,  
    ↳ MotoShieldObject.MOTO_YPIN1, MotoShieldObject.MOTO_YPIN2,  
    ↳ MotoShieldObject.PPR);  
setDirection(MotoShieldObject);  
end
```

Ďalšia metóda je `setDirection`, pomocou ktorej užívateľ nastavuje smer otáčania motora. Metóda očakáva jeden vstupný argument, ktorého hodnota je priradená premennej `direction`. V prípade, že je metóda zvolaná bez uvedenia argumentu, premennej bude priradená predvolená, kladná hodnota.

```
function setDirection(MotoShieldObject, direction)  
    if nargin < 2  
        direction = true;  
    end  
    MotoShieldObject.direction=direction;  
end
```

Metódy na zápis akčného zásahu, `actuatorWrite` a `actuatorWriteVolt` vyzerajú rovnako ako pri API pre Arduino IDE. Najprv sa porovná hodnota vstupného argumentu, ktorá symbolizuje hodnotu akčného zásahu v percentách a prípadne sa nasýti hodnotou minimálneho činiteľa plnenia na naštartovanie motora. Potom, na základe zvoleného smeru otáčania, sa pomocou príkazu `writePWMDutyCycle`

vyšle PWM signál z uvedeného „pin-u“. Tento príkaz očakáva hodnotu činiteľa plnenia v rozsahu 0 až 1.

```
function actuatorWriteVolt(MotoShieldObject, voltValue)
if (MotoShieldObject.direction)
writePWMDutyCycle(MotoShieldObject.arduino, MotoShieldObject.MOTO_UPIN1,
→ (voltValue^2/5^2));
writePWMDutyCycle(MotoShieldObject.arduino, MotoShieldObject.MOTO_UPIN2, 0);
else
writePWMDutyCycle(MotoShieldObject.arduino, MotoShieldObject.MOTO_UPIN1, 0);
writePWMDutyCycle(MotoShieldObject.arduino, MotoShieldObject.MOTO_UPIN2,
→ (voltValue^2/5^2));
end
end
```

Metóda `actuatorWriteVolt` vyzerá rovnako. Jediným rozdielom je prevod hodnoty vstupného argumentu z kvadratického priemeru napäťa na činiteľ plnenia PWM signálu, podľa vzťahu v Rov. (1.1). Z tohto dôvodu definícia metódy nie je uvedená.

Metóda `referenceRead` zostala rovnaká ako pri API pre prvú verziu. Jedinou zmenou je nahradenie konštanty referencie napäťa premennou `VREF`, ktorej hodnota sa mení v závislosti od verzie prístroja v rámci konštruktora triedy.

Na meranie odberu prúdu máme metódu `sensorRead`. Pri tejto metóde pomocou príkazu `readVoltage` zistíme napätie na analógovom vstupe, A3. Následne nameranú hodnotu prevedieme na jednotku ampér podľa Rov. (2.2).

```
function current = sensorReadCurrent(MotoShieldObject)
current = readVoltage(MotoShieldObject.arduino,MotoShieldObject.MOTO_YPIN) / 100;
end
```

Meranie uhlovej rýchlosťi v rámci rozhrania je možné tromi metódami. Prvá metóda v poradí, `sensorReadRPM`, vráti nameranú hodnotu v jednotke  $\text{ot} \cdot \text{min}^{-1}$ . Druhá, `sensorRead` v percentách a posledná v jednotke  $\text{rad} \cdot \text{s}^{-1}$ . V každej z uvedených metód je meranie uhlovej rýchlosťi uskutočnené prostredníctvom objektu `encoder`, spustením metódy `readSpeed`. Metóda `readSpeed` vráti hodnotu uhlovej rýchlosťi v jednotke  $\text{ot} \cdot \text{min}^{-1}$ , ktorú si potom jednoduchým spôsobom prevedieme na hodnotu v želanej jednotke. Keďže sú metódy na meranie uhlovej rýchlosťi takmer rovnaké, uvedená bude iba definícia metódy `sensorReadRadian`.

```
function rpm = sensorReadRadian(MotoShieldObject)
    rpm = readSpeed(MotoShieldObject.encoder)/60*2*pi;
end
```

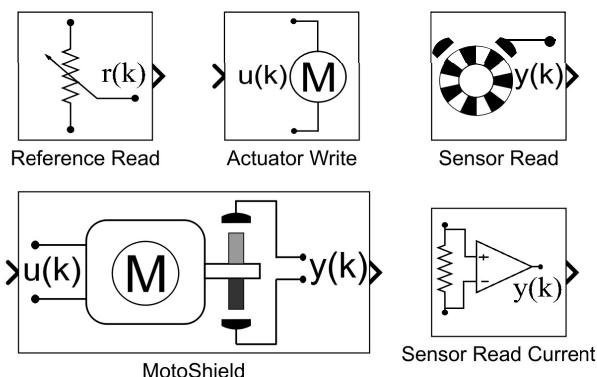
Kalibračná metóda `calibration` je rovnaká pre obidve verzie prístroja. Definíciu kalibračnej metódy s popisom sekvencie nájdete v bakalárskej práci [4].

## 3.4 API pre prostredie Simulink

Vývojové prostredie Simulink nám umožňuje pomocou grafického rozhrania vytvárať blokové schémy, pomocou ktorých sme schopný vykonávať simuláciu matematických modelov v čase. Na programovanie mikroradičových prototypizačných dosiek rodiny Arduino v prostredí Simulink je nám k dispozícii balík *Simulink Support Package for Arduino Hardware*. V rámci projektu *AutomationShield* bola napísaná knižnica, ktorá obsahuje programátorské rozhranie pre jednotlivé didaktické prístroje. Programátorské rozhranie sa skladá z jednotlivých blokov, resp. maskovaných subsystémov.

Pre prvú verziu prístroja bolo napísané programátorské rozhranie, ktoré obsahovalo nasledovné bloky:

- Reference Read - meranie referenčnej polohy potenciometra,
- Sensor Read - meranie uhlovej rýchlosťi motora,
- Sensor Read Current - meranie odberu el. prúdu motora,
- Actuator Write - vysielanie akčného zásahu,
- MotoShield - blok systému (ovládanie aj vstupov aj výstupov).



Obr. 3.1: Sada blokov Simulink API pre prístroj MotoShield.

V rámci tejto podkapitoly sa budeme snažiť spomenuté bloky rozšíriť tak, aby užívateľ bol schopný prostredníctvom masky nakonfigurovať daný blok bez ohľadu na verziu prístroja. Maska bloku je grafické okno, ktoré sa otvorí po dvojitom kliknutí na blok. Maska obsahuje interaktívne prvky ako sú napr. textové polia, tlačítka, padajúce menu a podobne. Pomocou interaktívnych prvkov sme schopný meniť hodnotu vnútorných parametrov a dokonca zmeniť aj vnútornú štruktúru bloku.

Voľbu prístroja užívateľovi umožníme v každom bloku prostredníctvom padajúceho menu s popisom „Shield Release“. Hodnota tohto prvku bude zaznamenaná v premennej `ShieldRelease`, ktorá môže nadobudnúť hodnoty R1 alebo R2.

### 3.4.1 Blok Reference Read

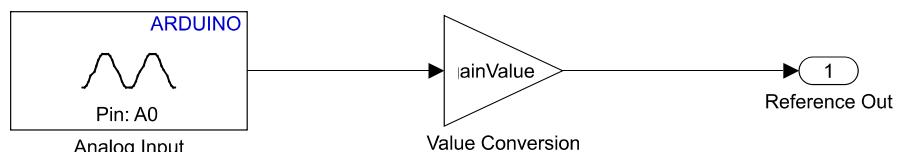
Ako bolo už spomenuté, úlohou bloku Reference Read je vyčítanie polohy potenciometra. Samotné vyčítanie hodnoty z AD prevodníka umožňuje blok **Analog Input**. Tento blok v rámci masky vyžaduje uvedenie čísla analógového vstupu, z ktorého chceme hodnotu čítať a taktiež aj hodnotu vzorkovacej periody. Hodnota vzorkovacej periody je prenesená z masky bloku Reference Read, kde je zapisovaná do textového poľa s premennou **Ts**. Číslo vstupu nie je rovnaké pre prvú a druhú verziu prístroja. Na základe voľby používaneho prístroja **ShieldRelease** sa v rámci tzv. callback funkcie prepisuje hodnota premennej **pinNumber**. Táto predstavuje číslo analógového vstupu potenciometra. Callback funkcia, ktorá nám toto umožní je nasledovná:

```
CurrentBlock = gcbh;
if get_param(CurrentBlock, 'ShieldRelease') == 'R1'
    set_param(gcb, 'pinNumber', num2str(4));
elseif get_param(CurrentBlock, 'ShieldRelease') == 'R2'
    set_param(gcb, 'pinNumber', num2str(0));
end
```

Užívateľ si má možnosť zvoliť jednotku v ktorej hodnota bude vystupovať. Voľbu jednotky umožňuje tzv. „radio“ tlačidlo s popisom „Readout Type“, kde prvá možnosť je hodnota v percentách, druhá je vo voltoch a posledná je hodnota z AD prevodníka. Zvolená možnosť tohto menu je uložená v premennej **radio**. Na základe tejto hodnoty je nastavená hodnota zosilnenia signálu v rámci inicializácie bloku nasledovne:

```
if radio==1
gainValue = (100/1023);
end
if radio==2 && strcmp(ShieldRelease, 'R1')
gainValue = (5/1023);
end
if radio==2 && strcmp(ShieldRelease, 'R2')
gainValue = (3.3/1023);
end
if radio==3
gainValue = 1;
end
```

Vnútornú štruktúru bloku vidíme na Obr. 3.2.



Obr. 3.2: Vnútorná štruktúra bloku Reference Read.

### 3.4.2 Blok Actuator Write

Na rozdiel od bloku Reference Read, vnútorná štruktúra bloku Actuator Write si odlišuje pre prvú a druhú verziu prístroja. Preto sme museli vnútornú štruktúru bloku rozdeliť na dva podsystémy, `ActuatorWriteR1` a `ActuatorWriteR2`. Vnútornú štruktúru bloku `ActuatorWriteR1` nájdete v bakalárskej práci [4].

Kedže pri komplilácii dochádzalo k chybám z dôvodu, že sa objavujú dva bloky na vysielanie PWM signálu s rovnakým „pin-om“, boli sme prinútení „zakomentovať“ podsystém v závislosti od voľby verzie prístroja. Komentovanie je uskutočnené v momente keď si užívateľ zvolí verziu prístroja v rámci callback funkcie prvku `ShieldRelease`. Komentovanie je uskutočnené pomocou nasledovného skriptu.

```
CurrentBlock = gcbh;
if get_param(CurrentBlock,'ShieldRelease') == 'R1'
    set_param(append(gcb,'/ActuatorWriteR1'),'Commented','off')
    set_param(append(gcb,'/ActuatorWriteR2'),'Commented','on')
elseif get_param(CurrentBlock,'ShieldRelease') == 'R2'
    set_param(append(gcb,'/ActuatorWriteR1'),'Commented','on')
    set_param(append(gcb,'/ActuatorWriteR2'),'Commented','off')
end
```

Premenná `gcb` je relatívna cesta aktuálneho bloku a pomocou funkcie `append` spojíme hodnoty premenných dátového typu `string`.

Dalej, užívateľ si môže zvoliť aj jednotku vstupného signálu, resp. akčného zásahu. Prvou z ponúknutých možností je akčný zásah v percentách, potom vo voltoch a na koniec v podobe 8-bitového<sup>4</sup> čísla. Vstupný signál je na začiatku orezávaný pomocou bloku `Saturation`. Potom na základe zvoleného vstupu je pomocou bloku `Multiport Switch` prepustený na bloky PWM. V prípade, že si užívateľ zvolil uviesť signál v percentách, vstupný signál je násobený číslom 2.55, keďže maximálna hodnota činiteľa plnenia pre blok PWM je 255. V prípade, že vstupný signál je vo voltoch, signál je upravený podľa Rov. (1.1).

Smer otáčania motora si taktiež užívateľ vie definovať v rámci masky, presnejšie pomocou padajúceho menu. Voľba je zaznamenaná v podobe poradového čísla do premennej `direction`. Blok `Variant Sink` nám umožňuje dynamický prepínať, na ktorý výstup pošleme vstupný signál. V prípade, že premenná `direction` nadobúda hodnotu 1, akčný zásah bude vysielaný cez vývod D5 a motor sa začne točiť v kladnom smere. V opačnom prípade bude akčný zásah vysielaný cez D6 a motor sa bude točiť v zápornom smere otáčania.

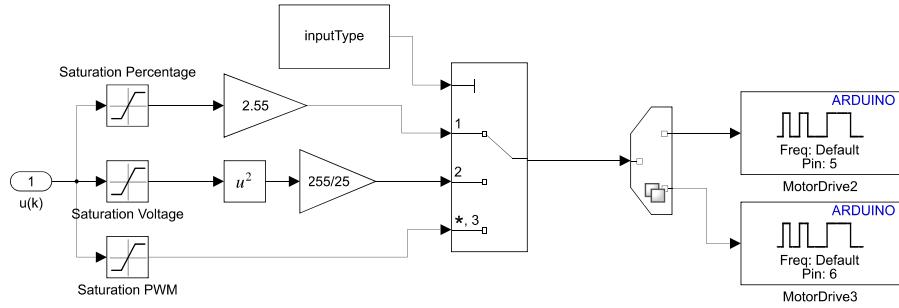
Vnútornú štruktúru bloku Actuator Write vidíme na Obr. 3.3.

### 3.4.3 Blok Sensor Read Current

Pomocou bloku Sensor Read Current je užívateľovi umožnené čítanie hodnoty odberu el. prúdu na motore. Na meranie prúdu pri obidvoch verziach prístroja je použitý rovnaký analógový vstup, A3. Jediným rozdielom je zosilnenie signálu. Zosilnenie výstupného signálu závisí od verzie prístroja. Pri prvej verzii je referenčné napätie na AD prevodníku 5 V a pri druhej verzii 3.3 V.

---

<sup>4</sup>Celočíselná hodnota v rozsahu 0 až 255.



Obr. 3.3: Vnútorná štruktúra subsystému `ActuatorWriteR2`, bloku `Actuator Write`.

V rámci masky je užívateľ schopný si nastaviť jednotku výstupnej veličiny. Ponúknuté možnosti sú ampér a miliampér. Informácia o zvolenej možnosti je zapísaná do premennej `unit`. Na základe tejto premennej a premennej `ShieldRelease` je určené zosilnenie výstupného signálu:

```

if (unit==1) && strcmp(ShieldRelease,'R1')
    gainValue=(5/1023/10)
end
if (unit==2) && strcmp(ShieldRelease,'R1')
    gainValue=(5/1023/10*1000)
end
if (unit==1) && strcmp(ShieldRelease,'R2')
    gainValue=(3.3/1023/100)
end
if (unit==2) && strcmp(ShieldRelease,'R2')
    gainValue=(3.3/1023/100*1000)
end

```

Vnútorná štruktúra bloku Sensor Read Current je rovnaká ako vnútorná štruktúra bloku Reference Read (Obr. 3.2), pričom je jediným rozdielom číslo „pin-u“.

### 3.4.4 Blok Sensor Read

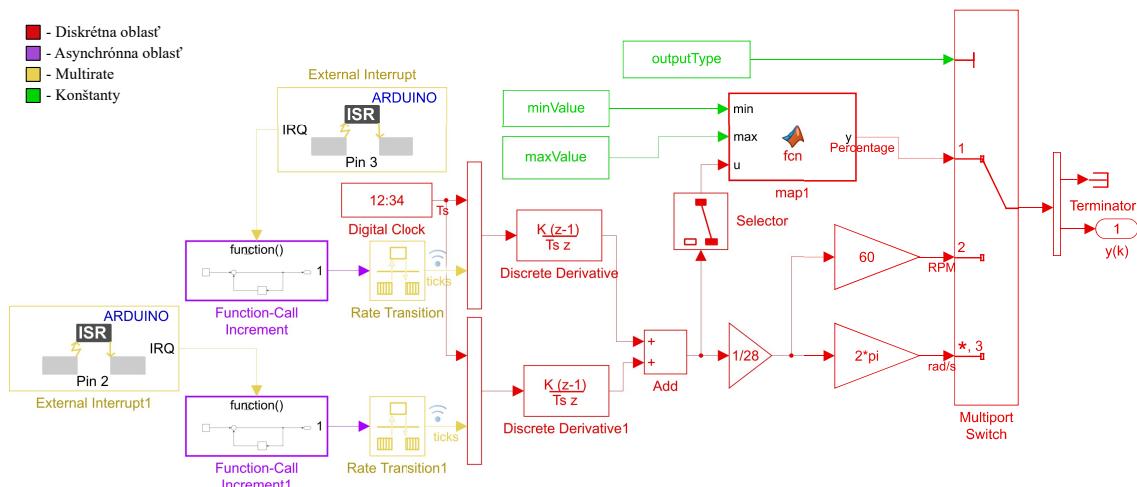
Na meranie otáčok za pomocí inkrementálneho snímača bol navrhnutý blok Sensor Read. Rovnako ako aj pri bloku Actuator Write, je vnútorná štruktúra bloku pre prvú a druhú verziu odlišná. Preto boli vytvorené dva subsystémy `SensorReadR1` a `SensorReadR2`, z ktorých je jeden zakomentovaný v závislosti od verzie prístroja. Komentovanie subsystému je realizované callback funkciou, rovnako ako pri bloku Actuator Write.

Externé prerušenie je nastavené pomocou blokov `External Interrupt` na digitálnych vstupoch D2 a D3. Výstupom z týchto blokov sú tzv. riadiace signály (angl. Control Signal), ktoré spúšťajú subsystémy `Funciton - Call Increment`. V spúštaných subsystémoch sa nachádza počítadlo. Výstupný signál zo subsystémov je asynchronný. Na to, aby sme previedli signál z asynchronnej do diskrétnej oblasti, potrebujeme zosynchronizovať pomocou časovaču, presnejšie pomocou bloku `Rate Transition` a bloku `Digital Clock` (Obr. 3.4). Synchronizovaný signál zatiaľ

nesie informáciu o uhle otočenia hriadeľa. Na výpočet uhlovej rýchlosťi signál zderivujeme. Kedžže pracujeme so signálom v diskrétnom čase, na deriváciu použijeme blok **Discrete Derivative**. Konečne, signál z výstupov D2 a D3 sčítame. Signál v tomto momente predstavuje počet impulzov inkrementálneho snímača za jednu sekundu. Na to aby sme signál vyjadrili v jednotke otáčka za sekundu, signál vydelíme číslom 28, keďže dostávame 28 impulzov za otáčku.

V prípade, že si užívateľ v maske zvolí výstup v jednotke radián za sekundu, signál ešte vynásobíme číslom  $2\pi$ . Otáčky za minútu dostaneme vynásobením číslom 60. Na vyjadrenie výstupného signálu v percentách musí užívateľ zadať maximálnu a minimálnu uhlovú rýchlosť motora ručne, alebo môže použiť prednastavené hodnoty. Preškálovanie signálu na percentá sa vykonáva v rámci bloku **map1**, presnejšie pomocou MATLAB funkcie **map**. Následne je hodnota signálu aj pre istotu orezaná, pomocou funkcie **constrain**. Uvedené funkcie boli napísané a prebraté zo zdrojového kódu projektu *AutomationShield*.

Na výstup je prepustený signál v závislosti od voľby jednotky, na základe premennej **outputType**, ktorá pôvodí z masky bloku.



Obr. 3.4: Vnútorná štruktúra subsystému **SensorReadR2**, bloku Sensor Read.

### 3.4.5 Blok MotoShield

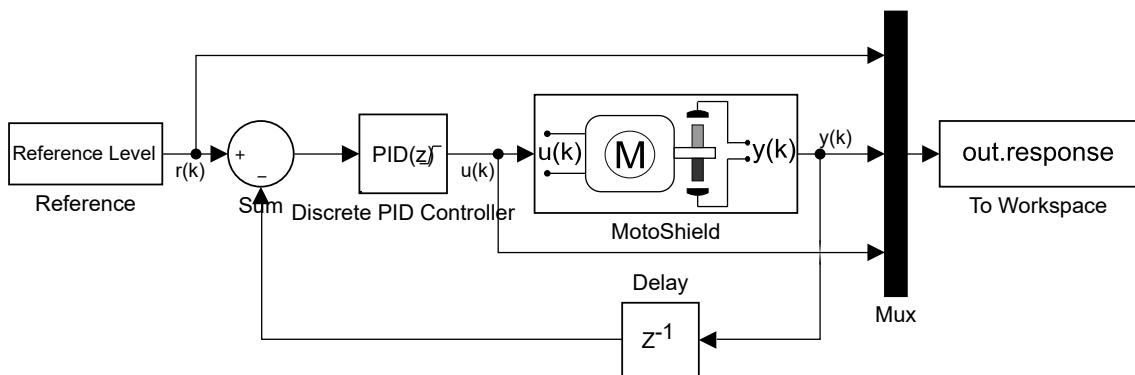
Blok MotoShield predstavuje celý systém, resp. súhrn všetkých doteraz spomenutých blokov. Pri tomto bloku je vstupná rovnako ako aj výstupná hodnota signálu v percentách. V rámci masky tohto bloku si užívateľ môže zvoliť verziu prístroja a smer otáčania motora. Taktiež si užívateľ môže vybrať, či hodnoty na preškálovanie výstupného signálu budú namerané pomocou kalibrácie alebo si ich užívateľ ručne zadá. Taktiež je v rámci masky od užívateľa žiadané, aby zadal vzorkovaciu periódu v sekundách. Na diagnostiku existuje aj možnosť si zvolať vnútornú referenciu, pričom je hodnota akčného zásahu určená polohou potenciometra.

V rámci bloku MotoShield, odber el. prúdu nie je meraný. V prípade, že by

užívateľ chcel merať odber prúdu, môže paralelne použiť blok Sensor Read Current v danom modeli.

Sekvencia na kalibráciu sa vykonáva pomocou MATLAB `function` bloku s menom `Calibration`. Bloky tohto typu počas simulácie spúšťajú program napísaný v jazyku MATLAB. Podstatným rozdielom v porovnaní s klasickým programom v MATLAB-e je, že v prípade Simulink-u je program spúštaný cyklicky, teda pri každom kroku simulácie. Preto v takomto prípade nie je vhodné použiť cyklus.

Voľba verzie je vyriešená, podobne ako v bloku Actuator Write, zakomentovaním jedného zo subsystémov `MotoShieldR1` a `MotoShieldR2`. Samotná štruktúra bloku je pomerne komplexná a preto nebude uvedená. Vnútornú štruktúru bloku ako aj funkciu na kalibráciu nájdete v MotoShield rozhraní pre Simulink. Programátorské rozhranie nájdete v zdrojovom kóde projektu `AutomationShield` [9]. Cesta k rozhraniu je nasledovná: „`AutomationShield/simulink/MotoLibrary.slx`“.



Obr. 3.5: Blok MotoShield zapojený v regolačnom obvode (prebraté z [4]).

Pri použití MotoShieldu v prostredí Simulink je dôležité nastaviť externý režim referencie na AD prevodníku. **Hardware Implementation, Hardware board settings, Analog input channel properties, External.** V opačnom prípade môže prísť k oškodeniu mikroradiiča.

# 4 Modelovanie, identifikácia a riadenie systému

V Kapitole 3 sme sa snažili vytvoriť programátorské rozhrania s cieľom, aby sme programátorovi zjednodušili prácu pri tvorbe ľubovoľných aplikácií. V tejto kapitole si ukážeme, ako použiť jednotlivé rozhrania na príkladoch týkajúcich sa teórie riadenia. Okrem iného budú popísané aj teoretické poznatky ku danej problematike. Na pochopenie funkcionálit, a implementáciu rozhrania pre dané vývojové prostredie, bude stačiť vysvetlenie jedného príkladu.

## 4.1 Matematicko-fyzikálna analýza

Systémy, s ktorými sa v technickej praxi stretávame, sme schopní popísať pomocou základných poznatkov z oblasti matematiky a fyziky. Najčastejšie systémy popisujeme s cieľom, aby sme poznali, resp. predikovali, ich správanie sa v budúcnosti, v závislosti od pozorovaných javov. Systémy popisujeme pomocou matematických rovníc, kde sú uvedené fyzikálne veličiny systému, ktoré sa navzájom ovplyvňujú. Postup odvodzovania rovníc sa nazýva *matematické modelovanie*, výsledkom ktorého je *matematický model* [4]. Pokiaľ nie sú časovo závislé, parametre modelu definujú vlastnosti, resp. kvantitatívne informácie o modelovanom systéme. Na to, aby sme určili parametre modelu, potrebujeme ich fyzický zmerať alebo môžeme pomocou tzv. experimentálnej identifikácie určiť ich hodnoty. Metóda experimentálnej identifikácie vyžaduje poznať hodnoty, resp. závislosť výstupných a vstupných veličín systému. Na základe týchto dát potom vieme, pomocou rôznych metód regresnej analýzy, určiť parametre systému tak, aby sa identifikovaný model čo najviac zhodoval s dátami.

Procesom experimentálnej identifikácie vieme jednoznačne určiť hodnotu hľadaného parametra iba v prípade, že počet neznámych parametrov je menší, ako počet rovníc matematického modelu. V opačnom prípade by sme vedeli určiť maximálne pomer hodnôt parametrov.

Pri modelovaní jednosmerného motora musíme vychádzať z tvrdenia, že je to systém schopný previesť elektrickú energiu na mechanickú. Preto budeme musieť popísať model aj z jednej, aj z druhej stránky. Najprv na základe druhového Newtonovho zákona pre rotačný pohyb vyjadríme výsledný točivý moment. Motor je poháňaný hnacím momentom  $\vec{M}_h$  a pri voľnom chode je zastavený pôsobením

trecích síl na dotykových plochách, resp. na ložiskách a na prevodovke.

$$J\ddot{\theta} = \sum_x \vec{M}_x = \vec{M}_h + \vec{M}_t \quad (4.1)$$

Ked'že je trenie sila pôsobiaca proti pohybu, v rovnici (4.2) bude trecí moment vychádzať  $\vec{M}_t$  so záporným znamienkom. Trecí moment je lineárne závislý od uhlovej rýchlosťi  $\dot{\theta}$  a konštanty viskózneho tlmenia  $b$ . Moment jednosmerného motora je úmerný odberu el. prúdu  $i$ , podľa konštanty  $K_t$ . Táto udáva informáciu o tom, koľko newtonmetrov dostaneme na hriadeľ za cenu jedného ampéra.

$$J\ddot{\theta} = M_h - M_t = K_t i - b\dot{\theta} \quad (4.2)$$

Ďalej, budeme vychádzať z druhého Kirchhoffovho zákona. Celkový úbytok napäťia na motore je spôsobený odporom  $R$ , indukčnosťou cievky  $L$  a elektromotorickou silou  $\mathcal{E}$ . Na základe Ohmovho zákona vieme, že úbytok napäťia je priamoúmerný el. prúdu a odporu. Ďalej, pri cievke je úbytok napäťia úmerný časovej zmene prúdu a elektromotorická sila, vzhľadom na úbytok napäťia, závisí od uhlovej rýchlosťi.

$$U_z = Ri + L\frac{di}{dt} + \mathcal{E} \quad (4.3)$$

$$U_z = Ri + L\frac{di}{dt} + K_e\dot{\theta} \quad (4.4)$$

V prípade, že by sme parametre  $K_t$  a  $K_e$  vyjadrili v základných jednotkách SI sústavy, zistili by sme, že sa jedná o rovnaký parameter. Ďalej ho budeme označovať symbolom  $K$  [18].

### Prenosová funkcia

Na základe diferenciálnych rovníc (4.2) a (4.3), pomocou Laplaceovej transformácie, vieme rovnice previesť z časovej, do obrazovej oblasti a následne ich scítat. Týmto dostaneme prenos systému v obrazovej oblasti.

$$G(s) = \frac{Y(s)}{U(s)} = \frac{s\Theta(s)}{U_z(s)} = \frac{K}{(K^2 + (Js + b)(Ls + R))} \quad (4.5)$$

Rov. (4.5) prenosu je prebratá zo záverečnej práce [4], kde je aj odvodnená.

### Lineárny stavový model

Stavový model je, rovnako ako prenosová funkcia a diferenciálna rovnica, jeden z typov reprezentácie matematického modelu, ktorým popisujeme reálny systém. Na rozdiel od prenosovej funkcie, v stavovom modeli definujeme medzi iným aj súvislost' jednotlivých stavov systému. Stavový model môžeme považovať za lineárny iba vtedy, keď sú všetky funkcie zmeny stavov a funkcie výstupov lineárne. Všeobecný stavový model je uvedený v Rov. (4.6), kde  $\mathbf{A}$  je matica dynamiky,  $\mathbf{B}$  je matica vstupov [16]. Zmenu stavu  $\mathbf{x}$  na základe aktuálneho stavu definuje matica  $\mathbf{A}$  a na základe vstupov, matica  $\mathbf{B}$ .

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{Ax}(t) + \mathbf{Bu}(t) \\ \mathbf{y}(t) &= \mathbf{Cx}(t) + \mathbf{Du}(t) \end{aligned} \quad (4.6)$$

Výstup modelu je definovaný pomocou matice výstupov **C**. Matica **D** je tzv. „Direct-feedthrough“ matica. Táto popisuje, v akom zmysle vstupné veličiny vplývajú priamo na výstupné. Pri väčšine systémov s ktorými sa stretávame, vstupné veličiny neovplyvňujú výstupné priamo, ale nepriamo. Preto je častokrát matica **D** nulová.

Pri našom modeli budeme uhlovú rýchlosť považovať za prvý stav a el. prúd za druhý stav systému. Tým pádom vektor stavov **x** bude mať nasledovný tvar:

$$\mathbf{x}(t) = \begin{bmatrix} \dot{\theta}(t) \\ i(t) \end{bmatrix} = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}. \quad (4.7)$$

Prvú rovnicu stavového modelu, alebo rovnicu dynamiky, vytvoríme tak, že do maticovej formy zapíšeme diferenciálnu Rov. (4.2) a Rov. (4.3).

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} -\frac{b}{J} & \frac{K}{J} \\ -\frac{K}{L} & -\frac{R}{L} \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ \frac{1}{L} \end{bmatrix} U_{\mathbf{z}}(t) \quad (4.8)$$

Druhá Rov. (4.9) stavového modelu je rovnica výstupu. Pomocou prístroja vieme merať aj uhlovú rýchlosť, aj odber prúdu a preto matica **C** bude jednotková, rozmeru  $2 \times 2$ . Matica **D** je pri našom systéme nulová.

$$\mathbf{y}(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x}(t) \quad (4.9)$$

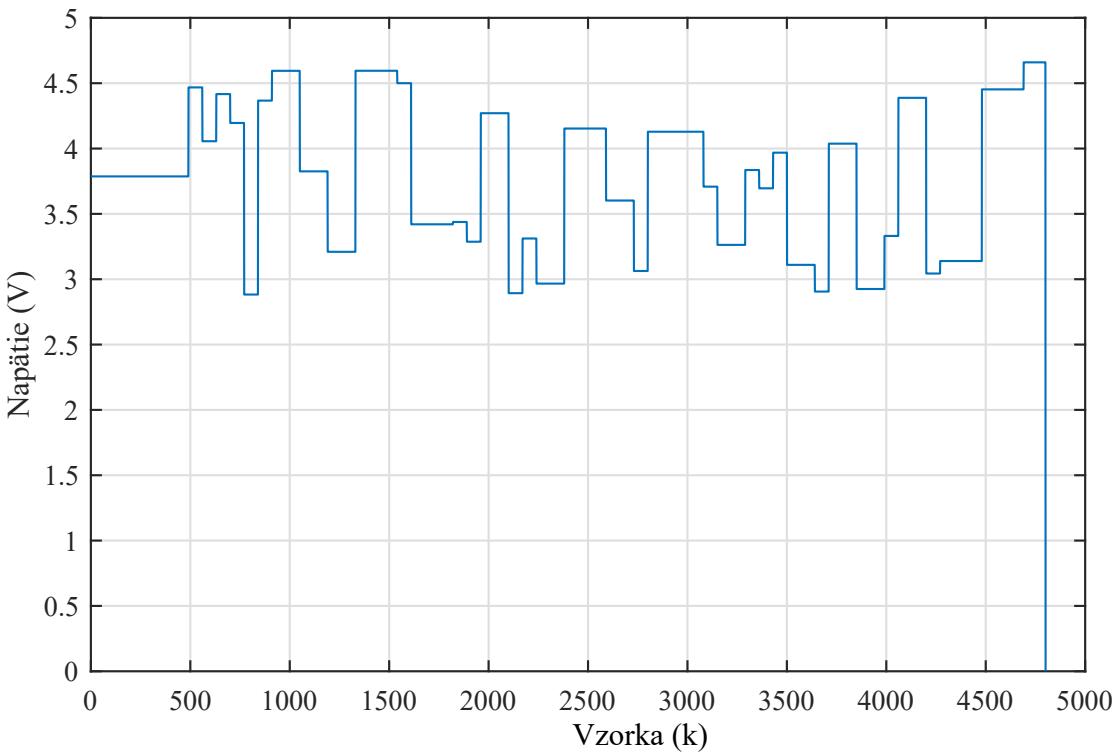
## 4.2 Identifikácia a odhad neznámych parametrov

Na identifikáciu parametrov pomocou experimentu potrebujeme poznať tzv. „Greybox“ model, ktorý sme si odvodili v predchádzajúcej časti v podobe prenosovej funkcie Rov. (4.5) a stavového modelu Rov. (4.8). Zostáva nám iba vykonať experiment, pomocou ktorého zistíme odozvu výstupných veličín na vstupnú. Na využitie odozvy potrebujeme testovací signál. Zvolíme si viac hladinový, pseudonáhodne vygenerovaný signál<sup>1</sup>. Signál sme vygenerovali pomocou MATLAB funkcie `aprbsGenerate`, ktorá bola napísaná v rámci knižnice projektu *AutomationShield* [9]. Priebeh vygenerovaného signálu vidíme na Obr. 4.1

Experiment sme vykonali pomocou príkladu `MotoShield_Identification`, ktorý bol napísaný v záverečnej práci [4]. Z experimentu sme zaznamenali hodnotu akčného zásahu, uhlovú rýchlosť a odber prúdu. Kedže je akčný zásah v podobe

---

<sup>1</sup>abbrev. APRBS.



Obr. 4.1: Viac hladinový pseudonáhodný testovací signál.

PWM signálu a impedancia motora je pomerne malá, nameraný signál odberu prúdu je zašumený a pripomína PWM signál. Skutočnú hodnotu odberu prúdu môžeme approximovať použitím hornopriepustného filtra, ktorý nám utlmí vysoké frekvencie signálu ale nízke ponechá. Bohužiaľ, tým pádom musí vzniknúť fázový posun vzhľadom na skutočný odber prúdu. Filtrovanie sme vykonali pomocou MATLAB funkcie `lowpass`<sup>2</sup>,

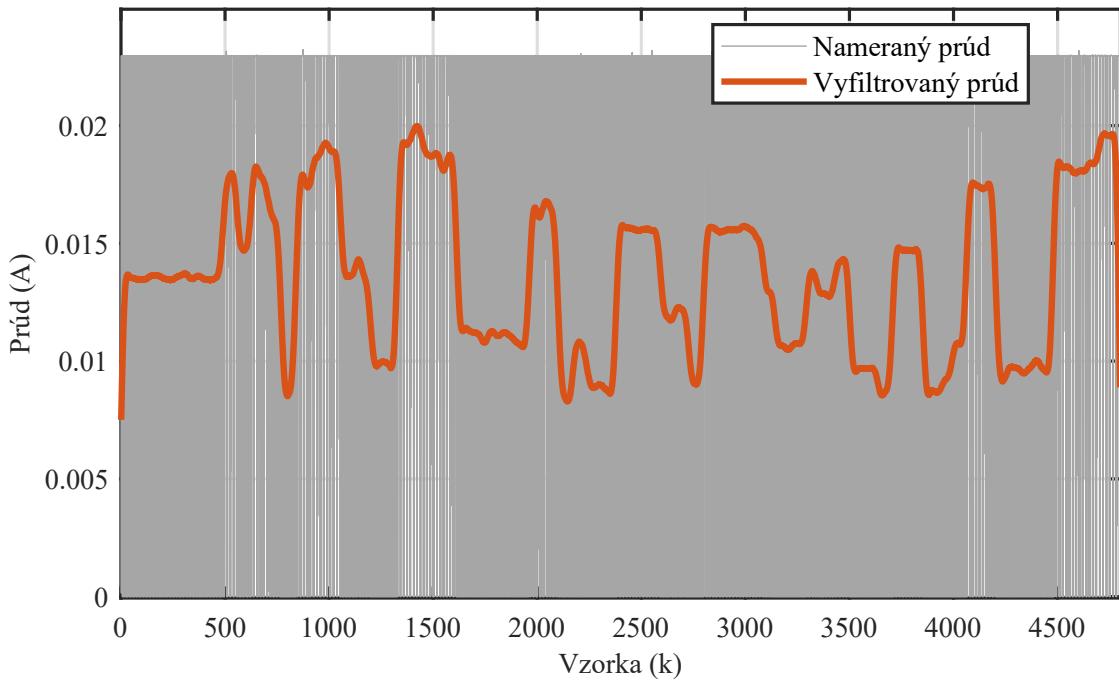
```
[ifltr,lp_filter]=lowpass(i,0.05,fs,'StopbandAttenuation',10,'Steepness',0.95);
```

kde sme v podobe argumentov nastavili parametre filtra. Použitý bol filter s konečnou odozvou, ktorý na rozdiel od filtra s nekonečnou odozvou zapríčiní menší fázový posun. Vyfiltrovaný signál odberu prúdu je porovnaný s nameraným na Obr. 4.2.

Na vykonanie identifikácie je potrebné zadefinovať si model, či už stavový pomocou príkazu `idss` alebo prenos pomocou `idtf`. Hodnotu el. odporu a indukčnosti cievky motora sme si namerali pomocou RLC metra. Hodnota odporu je  $10 \Omega$  a indukčnosť cievky je  $3.5^{-3} H$ . Hodnotu ostatných parametrov sme intuitívne odhadli. Taktiež sme pomocou vlasti `Free` vytvoreného objektu definovali, či daný parameter odhadujeme alebo nie. Pri prenosovej funkcie odhadujeme všetky parametre, kym pri stavovom modeli neodhadujeme iba parametre  $\frac{1}{L}$  a  $\frac{-R}{L}$ . Taktiež pomocou vlastností `Minimum` a `Maximum` sme definovali znamienko jednotlivých

---

<sup>2</sup>Súčasťou balíka: *Signal Processing Toolbox*.



Obr. 4.2: Porovnanie filtrovaného a nameraného signálu.

parametrov. Pomocou príkazu `iddata` sme vytvorili dátový objekt na identifikáciu. Konečné pomocou príkazov `tfest` a `ssest` sme vykonali identifikáciu modelu.

Porovnanie modelu s nameranými dátami pre prenosovú funkciu vidíme na Obr. 4.3. Porovnanie modelu s nameranými dátami pre stavový model vidíme na Obr. 4.4.

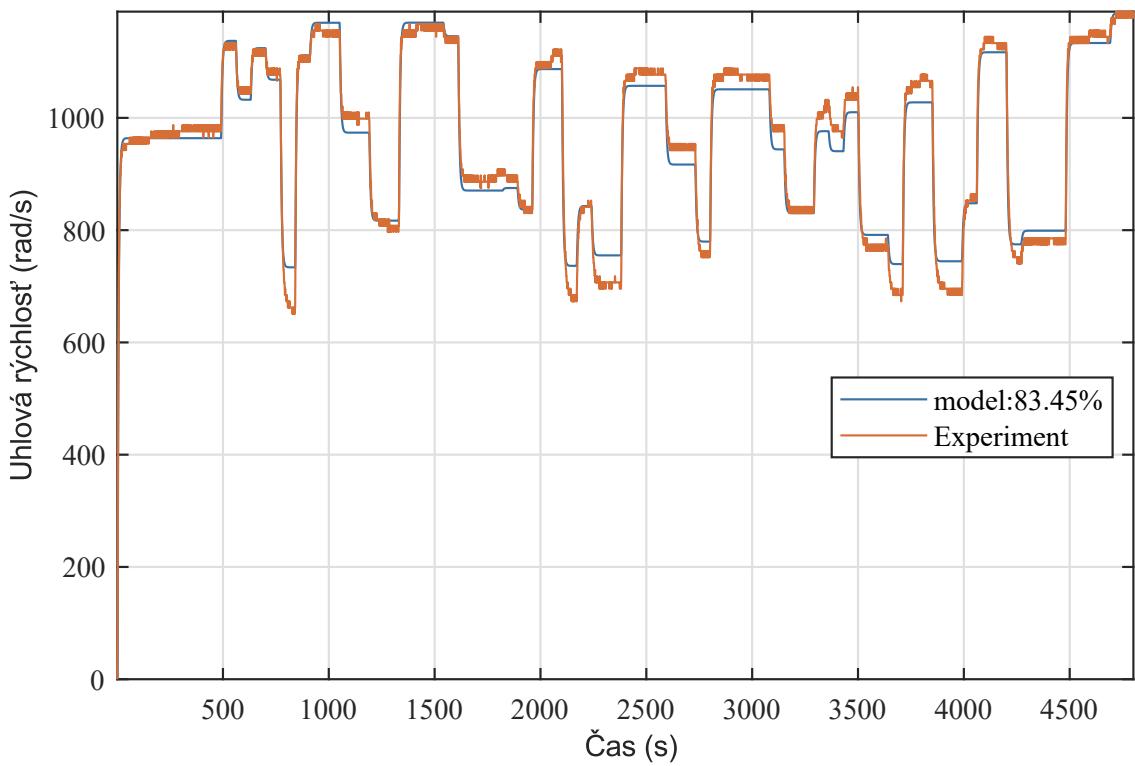
Identifikovaný model prenosovej funkcie je znázornený v Rov. (4.10) Identifikovaný stavový model je predstavený v Rov. (4.11).

$$G(s) = \frac{Y(s)}{U(s)} = \frac{123998}{s^2 + 57.41s + 487.2} \quad (4.10)$$

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} -11.0238 & 7.6432 \cdot 10^5 \\ -1.0769 & -2.7571 \cdot 10^3 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ 285.7143 \end{bmatrix} U_{\mathbf{z}}(t) \quad (4.11)$$

K hodnotám parametrov identifikovaného stavového modelu sa dostaneme spätným výpočtom:

$$\begin{aligned} L &= 0.0035 \text{ H} \\ R &= 10 \Omega \\ K &= 0.0038 \text{ Nm A}^{-1} \\ J &= 4.9312 \cdot 10^{-9} \text{ kg m}^{-2} \\ b &= 5.4360 \cdot 10^{-8} \text{ Nm s} \end{aligned}$$



Obr. 4.3: Porovnanie modelu prenosovej funkcie s dátami.

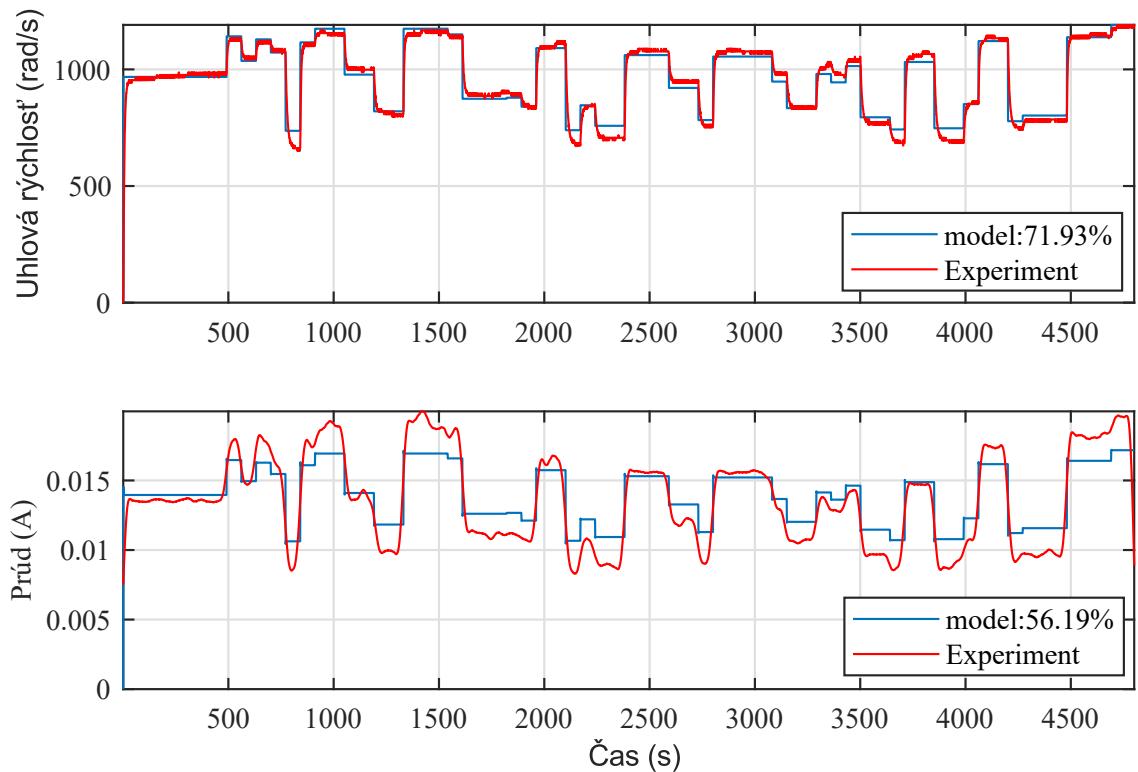
V rámci súborov MotoShield\_Identification\_SS a MotoShield\_Identification\_TF je kompletný skript na vykonanie identifikácie.

### 4.3 Odhad stavu

Verifikáciu stavového modelu môžeme urobiť aj pomocou Kalmanovho filtra. Kalmanov filter je algoritmus typu prediktor-korektor [16], pomocou ktorého sme schopní odhadnúť stav na základe modelu, výstupu a vstupu. Dôveryhodnosť modelu, resp. meraných dát, vieme nastaviť pomocou kovariančnej matice procesného  $\mathbf{Q}$  a kovariančnej matice šumu merania  $\mathbf{R}$ . V našom prípade je problematické meranie prúdu a preto kovariančné matice nadobúdajú nasledovné hodnoty:

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 \\ 0 & 0.01 \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix} \quad (4.12)$$

Samotnú simuláciu Kalmanovej filtrácie sme vykonali v rámci cyklu, pomocou funkcie `estimateKalmanState` [9], kde sme uviedli namerané vstupné a výstupné dátá, matice diskretizovaného identifikovaného stavového modelu a kovariančné matice. Na základe Obr. 4.5, vidíme, že filtrácia prúdu je dostatočne presná



Obr. 4.4: Porovnanie modelu stavového modelu s dátami.

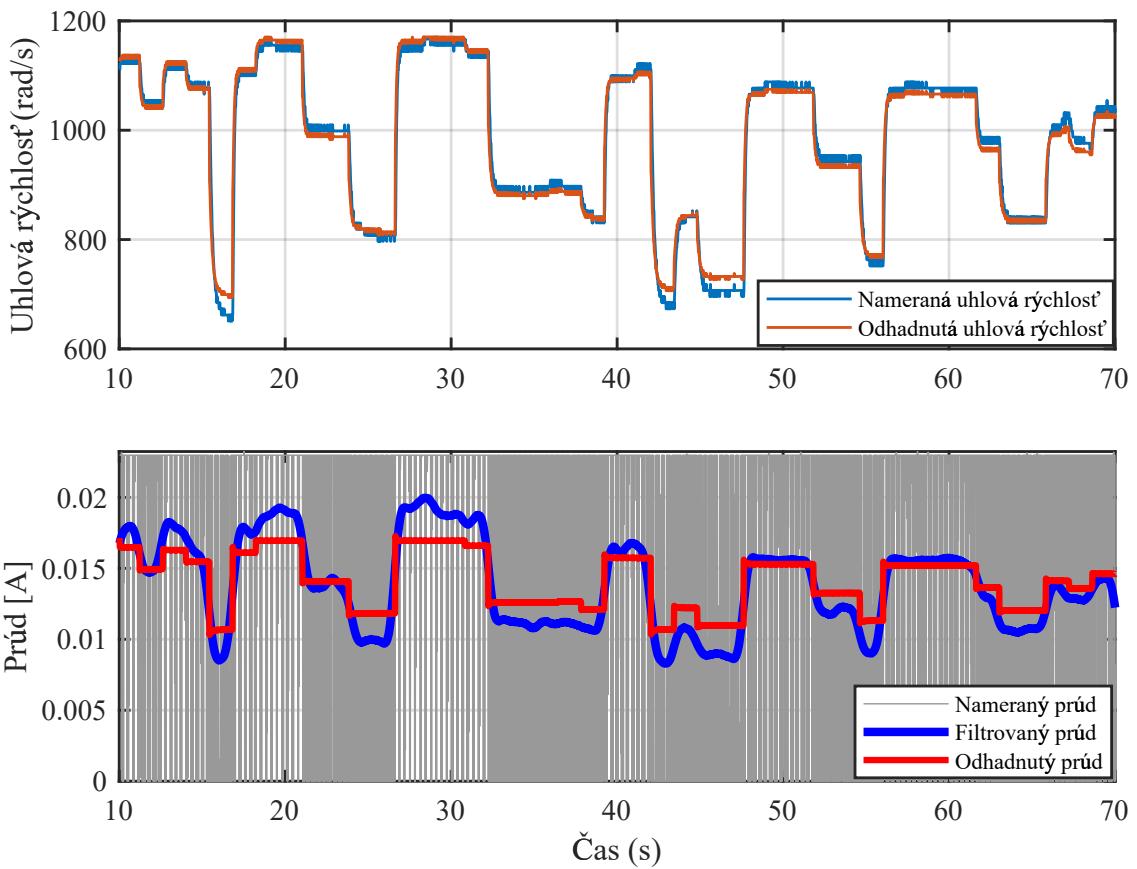
a tým pádom môžeme Kalmanov filter a identifikovaný model použiť na stavové riadenie. Viac o Kalmanovom odhadе stavu nájdete v knihe [16]. Skript na dosiahnuté výsledky nájdete v „/matlab/examples/MotoShield/MotoShield\_stateEstimate.m“.

## 4.4 Spätnoväzbové riadenie

Primárnym účelom prístroja MotoShield je výučba teórie riadenia. Pri systéme s jednosmerným motorom je najvhodnejšie použiť metódy riadenia so spätnou väzbou. V každom príklade riadenia budeme riadiť uhlovú rýchlosť hriadeľa. Kompenzátor a PID regulátor sú vstupno-výstupné typy regulátorov, čo znamená, že riadiť systém budeme iba na základe uhlovej rýchlosť vzhľadom na prenosovú funkciu. Pri LQ riadení poznáme prepojenie stavov, tým pádom jeden výstup vieme riadiť obidvoma stavmi.

### 4.4.1 Kompenzátor

Kompenzátor je regulátor v podobe prenosu s jedným pólom a jednou nulou. Ideou je ovplyvniť polohu núl a pôlov v z-rovine. Kompenzátor môže byť typu „Phase Lag“ alebo „Phase Lead“. „Lead“ kompenzátor je prenos, ktorého pól je číslo väčšie od nuly prenosu v oblasti obrazu. Inými slovami, nula je bližšie k imaginárnej osi od pôlu. Pri „Lag“ kompenzátore je situácia opačná [14].

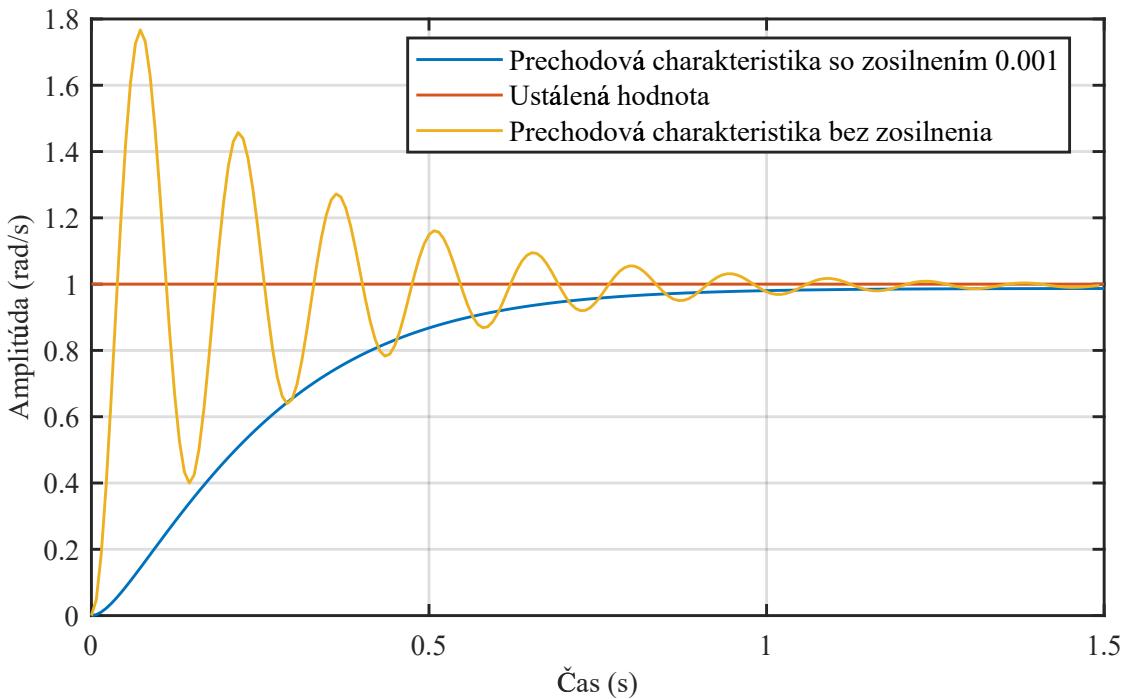


Obr. 4.5: Porovnanie výstupu a odhadu stavu na základe identifikovaného prenosu.

V našom prípade použijeme „Lag“ kompenzátor, keďže je schopný lepšie uriaodiť systém z pohľadu trvalej regulačnej odchýlky. Najjednoduchší spôsob ako navrhnúť takýto regulátor je podľa nasledovného postupu:

1. Identifikujeme prenosovú funkciu.
2. Vykreslíme odozvu na jednotkový skok.
3. V závislosti od odozvy, zvolíme zosilnenie. V prípade, že nastane prekmit, zosilnenie bude menšie ako 1. V opačnom prípade skúsime zmenšiť dobu nábehu zosilnením väčším ako 1.
4. Zadefinujeme prenos kompenzátoru v spojitom čase a zvolíme si hodnotu pôlu nuly tak, aby sme získali čo najrýchlejšiu odozvu bez prekmítu.
5. Regulačný obvod odsimulujeme.
6. Nakoniec prenos kompenzátoru diskretizujeme a odsimulujeme ho v uzavretej slučke s diskrétnym modelom.

V našom prípade, pri jednotkovom skoku, nastane prekmit a systém osciluje (Obr. 4.6), kym sa nedostane na ustálenú hodnotu. Z tohto dôvodu sme zvolili najmenšie zosilnenie, pri ktorom nepríde k prekmítu. Zvolené zosilnenie má hodnotu  $10^{-3}$ .



Obr. 4.6: Prechodová charakteristika prenosovej funkcie.

Následne sme zvolili hodnotu pôlu  $-0.05$  a hodnotu nuly  $-15$ . Prenos navrhnutého „Lag“ kompenzátoru:

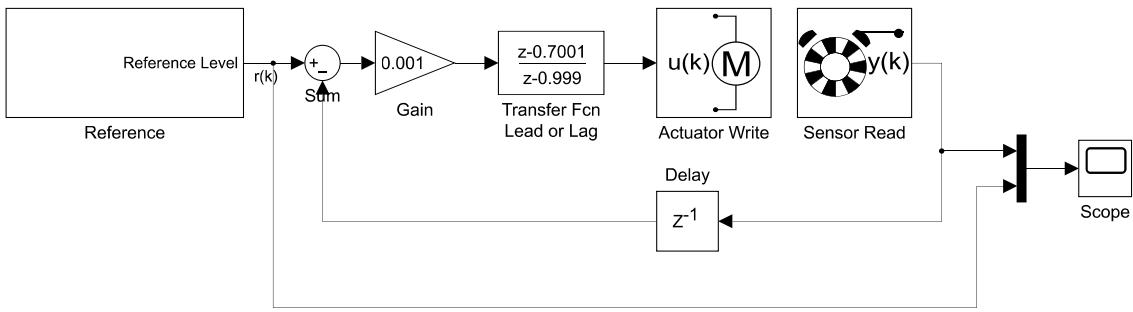
$$G(s) = 0.001 \frac{s + 15}{s + 0.05}. \quad (4.13)$$

Prenos kompenzátoru potrebujeme mať v diskrétnnej forme, čo vieme dostáť pomocou funkcie `c2d`, kde uvedieme prenos a vzorkovaciu períodu  $0.02$  s. Diskrétny prenos nášho „Lag“ kompenzátoru je nasledovný:

$$G(z) = \frac{0.001z - 0.0007}{z - 0.999} = 0.001 \frac{z - 0.7}{z - 0.999}. \quad (4.14)$$

Riadenie vykonáme v prostredí Simulink. Na vytvorenie regulačného obvodu (Obr. 4.7) potrebujeme blok na zápis akčného zásahu, Actuator Write, potom blok na meranie uhlovej rýchlosťi, Sensor Read a blok na vytvorenie signálu židanej hodnoty, Reference. Samozrejme, keďže ide o diskrétny obvod, potrebujeme oneskoríť výstup o jednu vzorku pomocou bloku Delay. Blok Transfer Fcn Lead or Lag predstavuje regulátor a blok Gain zosilnenie. V každom bloku musí byť nastavená rovnaká vzorkovacia períoda, v našom prípade  $T_s = 0.02$  s. Blok na meranie uhlovej rýchlosťi nastavíme tak, aby nám vrátil hodnotu v jednotke  $\text{rad} \cdot \text{s}^{-1}$ , podľa prenosovej funkcie, pomocou ktorej sme regulátor nalaďili. Hodnota akčného zásahu je rovnaká ako pri prenosovej funkcií, volt.

Regulačný pochod navrhnutého obvodu (Obr 4.8) je dostatočne dobrý, nedošlo k prekmitu a doba prechodu je dostatočne krátka.



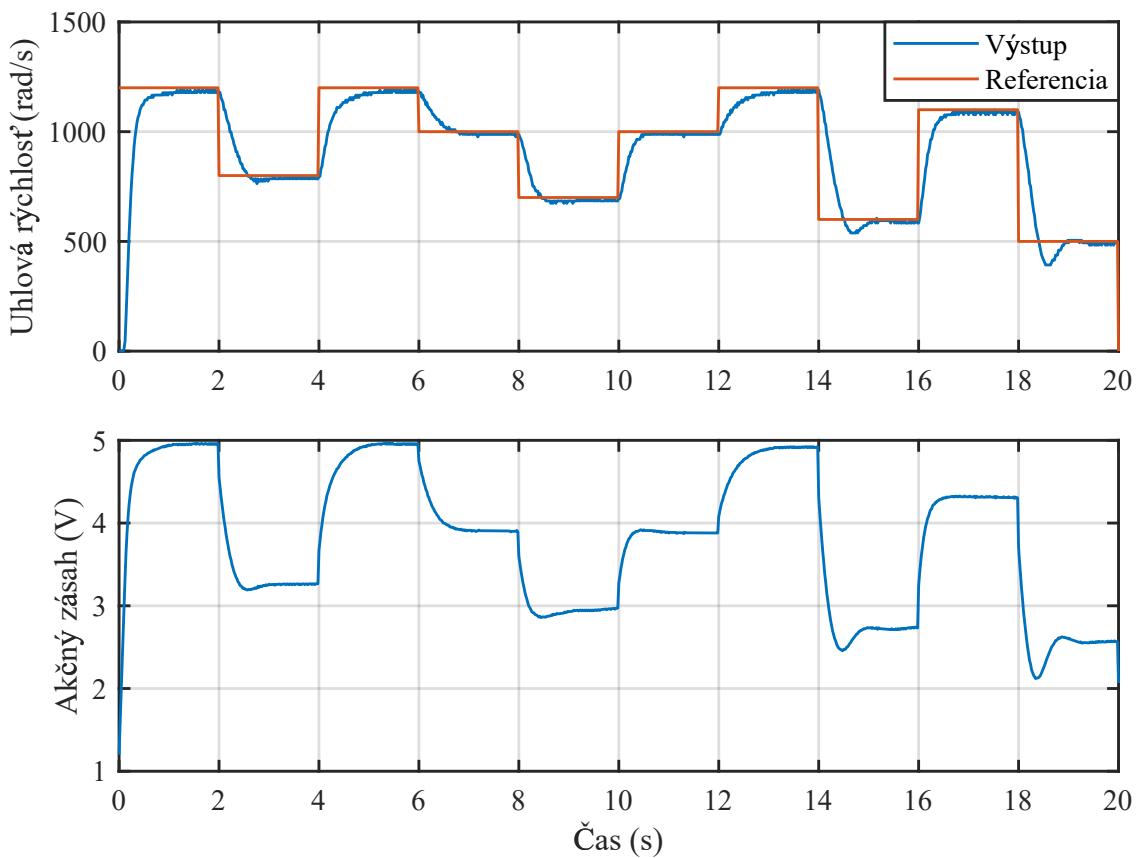
Obr. 4.7: Schéma riadiaceho obvodu „Lag“ kompenzátoru.

#### 4.4.2 PID riadenie - metóda Ziegler-Nichols

Jedným z najčastejšie používaných regulátorov v praxi je PID. Jedná sa o vstupno-výstupný regulátor s troma zložkami, súčtom ktorých je akčný zásah. Proporcionalná zložka je v zásade iba zosilnenie regulačnej odchýlky. Integračná zložka kumulatívne sčítuje hodnotu regulačnej odchýlky. Proporcionalná zložka nedokáže vynulovať trvalú regulačnú odchýlku. Na vynulovanie regulačnej odchýlky potrebujeme práve I zložku. Dobu prechodu vieme zmenšiť navýšením zložiek P a I. Avšak si treba dať pozor, pretože obidve tieto zložky zvyšujú pravdepodobnosť prekmitu. Vysoká hodnota zosilnenia I zložky môže mať za dôsledok predĺženie doby ustálenia a dokonca môže spôsobiť nestabilitu systému. Na elimináciu prekmitu je derivačná zložka. Derivačná zložka prináša vyššiu stabilitu regulačného obvodu tým, že zmenšuje akčný zásah na základe rýchlosťi zmeny regulačnej odchýlky [3].

PID regulátor má niekoľko typov zápisu. Keďže postupujeme podľa Ziegler-Nicholsovej metódy ladenia, použijeme absolútny PID regulátor. Taktiež použijeme aj saturáciu akčného zásahu a saturáciu integračnej zložky. Integračná zložka, v prípade, že sa výstup dlhšiu dobu z nejakého dôvodu nedostane k zadanej hodnote, sa nasýti, akčný zásah bude istú dobu maximálny. Toto môže spôsobiť haváriu systému alebo len pokazí proces regulácie. Preto integračnú zložku je potrebné saturovať, najčastejšie v intervale rozsahu akčného zásahu.

Pri Ziegler-Nicholsovej metóde vychádzame z prechodovej charakteristiky systému. Na získanie hodnôt parametrov potrebujeme vedieť dobu nábehu, dobu prieťahu a zosilnenie. Zosilnenie  $K$  je posledná hodnota výstupu prechodovej odozvy. Dobu nábehu  $T_n$  a dobu prieťahu  $T_u$  dostaneme pomocou dotyčnice v inflexnom bode prechodovej charakteristiky. Doba prieťahu je hodnota medzi nulovým bodom, v prípade, že je dopravné oneskorenie nulové a medzi bodom v ktorom dotyčnica pretína x-ovú os. Doba nábehu je od druhého bodu doby prieťahu až po bod keď dotyčnica nadobudne hodnotu zosilnenia  $K$  [3]. Inflexný bod funkcie nájdeme zistením maxima primitívnej funkcie. Inými slovami, numericky zderivujeme prechodovú charakteristiku a najdeme maximum. Rovnicu dotyčnice nájdeme pomocou všeobecného vzorca pre dotyčnicu, viď Rov. (4.15), v ktorom  $x_0$  a  $f(x_0)$  sú



Obr. 4.8: Regulačný pochod pri riadení pomocou „Lag“ kompenzátoru.

súradnice inflexného bodu.

$$\begin{aligned} y &= f(x) \\ y &= \frac{df(x_0)}{dx}(x - x_0) + f(x_0) \end{aligned} \quad (4.15)$$

Po vykreslení prechodovej charakteristike identifikovanej prenosovej funkcie s dotyčnicou, zistujeme, že  $T_n = 0.148$ ,  $T_u = 0.011$ ,  $K = 1271$ .

Podľa metódy Ziegler-Nichols, vypočítame parametre absolútneho PID regulátora nasledovne [3]:

$$K_p = 1.25 \frac{T_n}{KT_u} = 1.25 \frac{0.148}{1271 \cdot 0.011} = 0.0127 \quad (4.16)$$

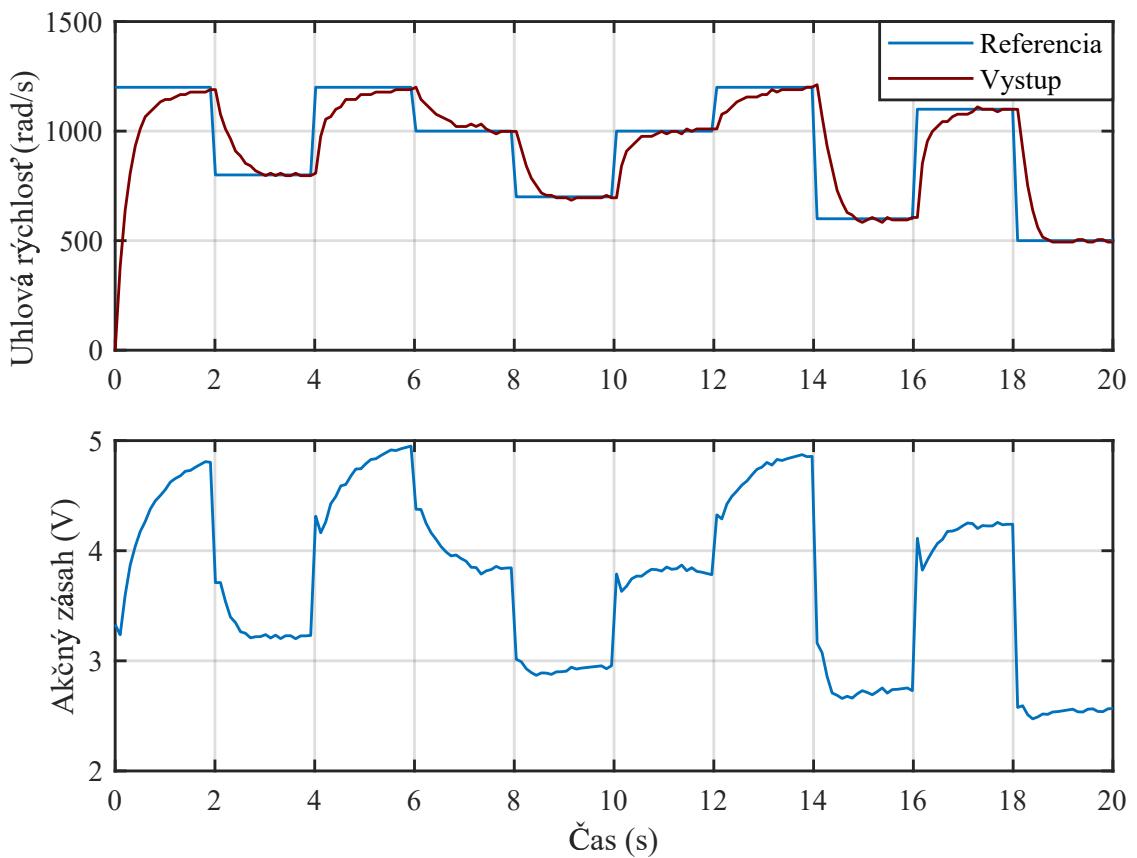
$$T_I = 2T_u = 2 \cdot 0.011 = 0.0221 \quad (4.17)$$

$$T_D = \frac{1}{2}T_u = 0.5 \cdot 0.011 = 0.0055 \quad (4.18)$$

Riadenie systému budeme realizovať v prostredí MATLAB. V záverečnej práci [4], autor navrhoval príklad pre PID riadenie. Konštanty regulátora študent navrhoval iba heruistickej a práve toto bola aj jedna z poznámok oponenta. Napísaný príklad si prispôsobíme pre náš regulátor, zmeníme hodnoty parametrov regulátora,

vzorkovaciu periódou a spustíme program. V uvedenej práci nájdete detailný popis príkladu.

Rovnako, ako v príklade pre „Lag“ kompenzátor, uhlová rýchlosť vystupuje v jednotke  $\text{rad} \cdot \text{s}^{-1}$  a akčný zásah vo voltoch. Preto je potrebné v príklade metódu `actuatorWrite` nahradíť metódou `actuatorWriteVolt` a `sensorRead` metódou `sensorReadRadian`. Vzorkovaciu periódou uvedieme rovnakú, ako v je diskrétnom prenose,  $T_s = 0.02$ . Po spustení príkladu dostaneme priebeh, znázornený na Obr. 4.9, kde vidíme, že použitá metóda ladenia PID regulátora sa ukázala ako vyhovujúca pre daný model, resp. systém.



Obr. 4.9: PID riadenie - Ziegler-Nicholsova metóda ladenia.

#### 4.4.3 Stavové riadenie

V porovnaní s vstupno-výstupnými regulátormi sú stavové regulátory komplexnejšie, z toho dôvodu, že sa spoliehajú na matematicko-fyzikálny model riadeného systému. Algoritmy stavového riadenia sú vhodné na použitie najmä pre systémy, ktorých vnútorné stavy sú navzájom prepojené. V stavovom modeli matica dynamiky nesie informáciu o prepojení stavov. Vidíme, že v prípade jednosmerného motora ani jeden člen matice na vedľajšej diagonále  $\mathbf{A}$  nie je nulový. To znamená, že obidva stavy sa navzájom ovplyvňujú. Z tohto dôvodu sa v tejto časti budeme snažiť navrhnúť riadiaci obvod so stavovým regulátorom.

Lineárny kvadratický regulátor (ang. Linear Quadratic<sup>3</sup> Controller) je jeden zo známejších, keď ide o stavové riadenie. Jedná sa o regulátor, ktorého zosilnenie, vzhľadom k stavovému modelu, je optimálne. Nedostatkom je, že rovnako ako aj vstupno-výstupné regulátory, LQ regulátor neberie ohľad na procesné ohraničenia systému. Tým pádom, pre takéto systémy nemôžeme LQ zosilnenie, resp. riadenie považovať za optimálne. Každopádne LQ regulátor je robustnejší od vstupno-výstupných regulátorov. Toto spočíva hlavne v tom, že sme schopní riadiť všetky merané alebo odhadnuté stavy systému súčasne. Keď sa pozrieme na riadiaci zákon pre riadenie s nulovou referenciou [16],

$$\mathbf{u}_k = -\mathbf{K} \left( \mathbf{x}_k - \mathbf{r}_k \right), \quad (4.19)$$

kde,

- $\mathbf{K}$  je vektor optimálneho zosilnenia,
- $\mathbf{x}_k$  je vektor stavov v diskrétnom čase,
- $\mathbf{r}_k$  je vektor žiadanych hodnôt v diskrétnom čase,
- $\mathbf{u}_k$  je vektor akčného zásahu v diskrétnom čase,

vidíme, že referenčná hodnota je definovaná v podobe vektora s rovnakým rozmerom ako je vektor stavov. To znamená, že sme schopní zadefinovať žiadanú hodnotu pre každý stav a vypočítať regulačnú odchýlku riadených stavov.

Optimálne zosilnenie  $\mathbf{K}$  je také, ktoré minimalizuje indikátor kvality  $J$  kvadratickej účelovej funkcie stavov a vstupov systému [16]:

$$\mathbf{J}_k(\mathbf{x}, \mathbf{u}) = \sum_{k=1}^{\infty} \left( \mathbf{x}_k^T \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k \right), \quad (4.20)$$

pričom uvažujeme riadiaci zákon LQ s nulovou žiadanou hodnotou. K vyriešeniu optimalizačnej úlohy, resp. vypočítaniu optimálneho zosilnenia, sa dostaneme riešením Riccatiho rovnice [16]

$$\mathbf{P}_k = \mathbf{A}^T \mathbf{P}_k \mathbf{A} - \mathbf{A}^T \mathbf{P}_k \mathbf{B} \left( \mathbf{R} + \mathbf{B}^T \mathbf{P}_k \mathbf{B} \right)^{-1} \mathbf{B}^T \mathbf{P}_k \mathbf{A} + \mathbf{Q} \quad \text{kde,} \quad (4.21)$$

$$\mathbf{K} = \left( \mathbf{R} + \mathbf{B}^T \mathbf{P}_k \mathbf{B} \right)^{-1} \mathbf{B}^T \mathbf{P}_k \mathbf{A}. \quad (4.22)$$

Matica  $\mathbf{P}$  je matica penalizácie koncového stavu.

Diagonálne matice  $\mathbf{Q}$  a  $\mathbf{R}$  sú penalizačné matice stavov a vstupov. Pomocou týchto matíc sme schopní určiť aký vplyv budú mať stavy, resp. vstupy pri riadení [16]. Na určenie hodnoty penalizačných matíc musíme poznáť riadený systém a spoľahlivosť modelu. V prípade, že niektorý zo stavov nevieme kvalitne merať, odhadnúť alebo je menej dôležitý pre úlohu riadenia, jemu príslušná hodnota v matici  $\mathbf{Q}$  bude pomerne menšie číslo vzhľadom na ostatné. Pri penalizácii akčných členov

---

<sup>3</sup>abbrev. LQ.

má číselná hodnota opačný význam. Čím je číslo menšie, tým viac je uprednostnený daný vstup. V prípade, že v systéme máme viac akčných členov môžeme ich penalizáciu určiť na základe rôznych vlastností. Ako napr. nízka kvalita aktuátora, vysoká spotreba energie, pomalá dynamika a podobne. Penalizačné matice a kvalita modelu sú najdôležitejšie body pri návrhu LQ regulátora. Pri riadení systémov, ktoré nestabilným regulovaním vieme dovest' do havarijného stavu, je odporúčané najprv zvoliť vysokú penalizáciu vstupov a postupne penalizáciu znižovať. Taktiež môžeme regulačný obvod najprv odsimulovať alebo zistiť vlastne čísla uzavretej slučky LQ regulátora príkazom `eig(A-B*K)` v MATLAB-e [16]. V prípade, že sme v diskrétnom čase a absolútnej hodnote reálnych zložiek vlastných čísel je menšia ako 1, riadiaci obvod považujeme za stabilný.

LQ regulátor nepozná regulačnú odchýlku v minulosti, spolieha sa iba na zosilnenie na aktuálnej regulačnej odchýlke. Toto väčšinou spôsobuje trvalú regulačnú odchýlku, keďže model takmer nikdy nie je dostatočne presný aby sme sa tomuto vyhli. Preto na vynulovanie trvalej regulačnej odchýlky pridáme integračnú zložku v podobe ďalšieho (fiktívneho) stavu  $\mathbf{x}^I$ . Integračný stav vypočítame kumulatívne

$$\mathbf{x}_{k+1}^I = \mathbf{x}_k^I + (\mathbf{r}_k - \mathbf{Cx}_k) \text{ kde,} \quad (4.23)$$

$$\mathbf{Cx}_k = \mathbf{y}_k, \quad (4.24)$$

v prípade, že matica  $\mathbf{D}$  je nulová. Na to aby sme vypočítali optimálne zosilnenie s uvažovaním integračnej zložky, Rov. (4.23) zapíšeme do rovnice dynamiky stavového modelu:

$$\begin{bmatrix} \mathbf{x}_{k+1} \\ \mathbf{x}_{k+1}^I \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ -\mathbf{C} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{x}_k^I \end{bmatrix} + \begin{bmatrix} \mathbf{B} \\ \mathbf{0} \end{bmatrix} \mathbf{u}_k + \begin{bmatrix} \mathbf{0} \\ \mathbf{I} \end{bmatrix} \mathbf{r}_k. \quad (4.25)$$

Zosilnenie regulátora sme si vypočítali v MATLAB-e na základe direktného identifikovaného modelu. Najprv si stavový model rozšírime o integračný stav

```
Ts=0.02; % Vzorkovacia perioda
zmodel = c2d(model,Ts); % Diskretizacia modelu
intA = [zmodel.A, zeros(2, 1); -zmodel.C(1,:), ones(1,1)];
intB = [zmodel.B; 0];
intC = [zmodel.C(1,:), 0];
```

Potom definujeme penalizačné matice a zároveň vypočítame optimálne zosilnenie príkazom `dlqr`. Jednotkové matice sú dobrým počiatočným odhadom.

```
Q_lqr=diag([1 1 1]); % penalizacia stavov
R_lqr=1; % Penalizacia vstupov
K=lqr(intA,intB,Q_lqr,R_lqr,Ts)
```

Prvý spôsob akým vieme otestovať stabilitu je výpočtom vlastných čísel regulačného obvodu.

```
>> eig(intA-intB*K)
```

```
ans =
-57.4694
-1.4149
-0.0002
```

Na základe prvého a druhého vlastného čísla vidíme, že regulačný obvod nie je stabilný a preto budeme penalizovať aktuátor vysokým číslom  $R_{lqr}=1e8$ . Výpočet zosilnenia a kontrolu stability vykonáme znova.

```
>> K=lqrdf(intA,intB,Q_lqr,R_lqr,Ts)
```

```
>> eig(intA-intB*K)
```

```
K =
0.0079    0.0014   -0.0080
ans =
-0.9791
-0.0361
-0.0002
```

Absolútna hodnota všetkých vlastných čísel je menšia ako 1 a tým pádom regulačný obvod môžeme považovať za stabilný.

LQ riadenie sme na zariadení vykonali cez prostredie Arduino IDE. Sem sme museli zahrnúť aj odhad druhého stavu (prúdu), pretože je zašumený signál pri riadení nepoužiteľný. Kalmanov filter sme implementovali funkciou `getKalmanEstimate`, ktorý je prebratý zo zdrojového kódu projektu *AutomationShield*.

Najprv sme si zadefinovali vektor žiadanej hodnoty, matice stavového modelu a kovariančné matice a vektor počiatočného stavu pre Kalmanov filter [9].

```
float R[] =
→ {1200.0, 800.0, 1200.0, 1000.0, 700.0, 1000.0, 1200.0, 600.0, 1100.0, 500.0, 0.0};
BLA::Matrix<2, 2> A = {0.0013, 0.4064, 0, -0.0002};
BLA::Matrix<2, 1> B = {255.1968, 0.0038};
BLA::Matrix<2, 2> C = {1, 0, 0, 1};
BLA::Matrix<2, 2> Q_Kalman = {100, 0, 0, 0.01};
BLA::Matrix<2, 2> R_Kalman = {1, 0, 0, 10};
BLA::Matrix<2, 1> xIC = {0, 0};
```

Potom zadefinujeme vektor stavov (vrátane integračného stavu), vektor LQ zosilnenia a vektor referenčnej hodnoty.

Následne, v rámci cyklu vzorkovania s períodou  $T = 0.02s$  je spúšťaný riadiaci algoritmus a vykreslované sú hodnoty prostredníctvom sériovej komunikácie.

```
Xr(0) = R[i];
u = -(K*(X-Xr))(0); // Riadiaci zakon LQ
MotoShield.actuatorWriteVolt(u);
// Meranie
BLA::Matrix<2, 1> Y = {MotoShield.sensorReadRadian(),
→ MotoShield.sensorReadCurrent()};
// Odhad stavu
MotoShield.getKalmanEstimate(X, u, Y, A, B, C, Q_Kalman, R_Kalman, xIC);
X(2) = X(2) + (Xr(0) - X(0)); // Integrovana regulacna odchylka
```

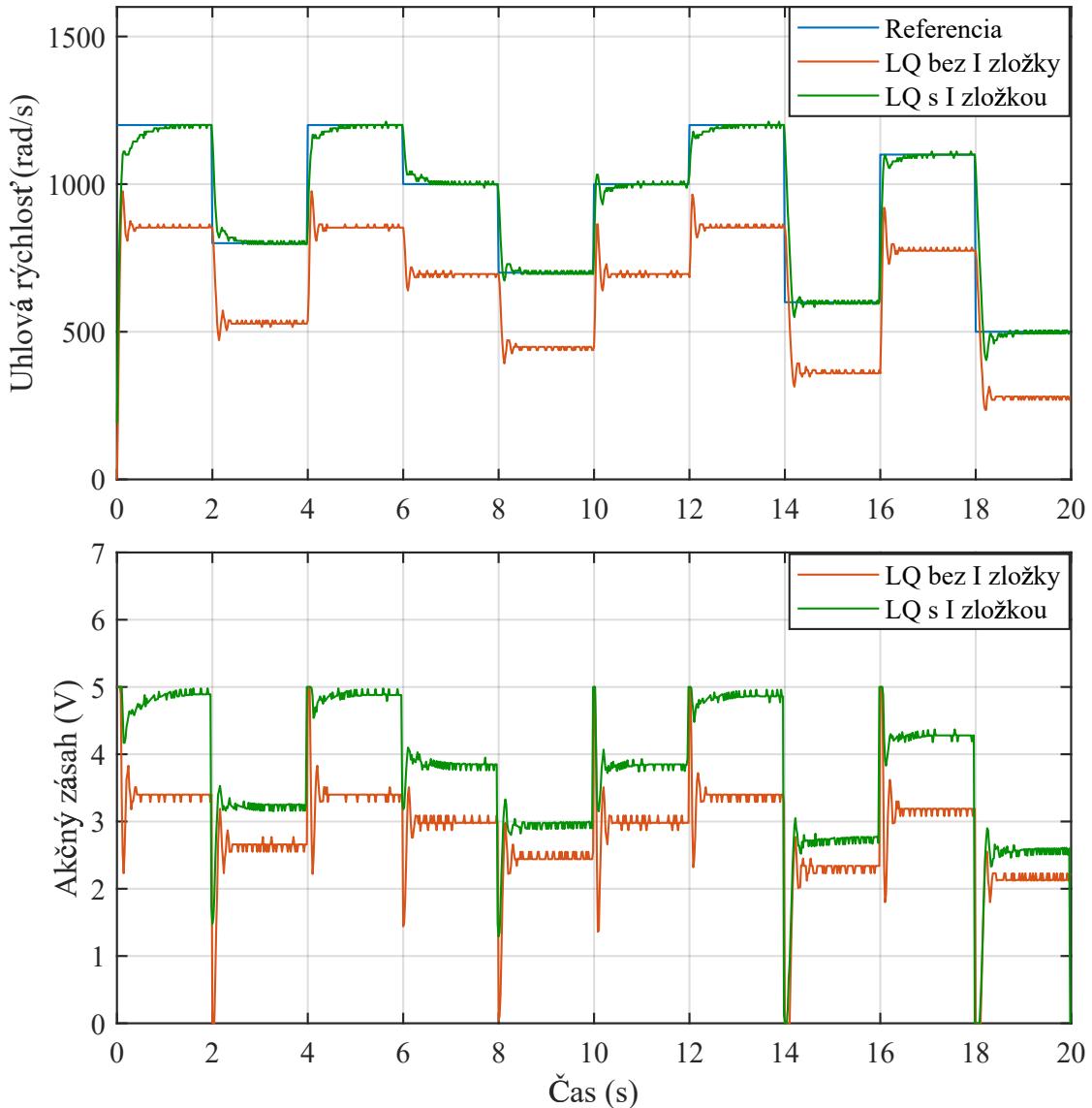
```

Serial.print(Xr(0));      // Referencia
Serial.print(", ");
Serial.print(Y(0));      // Vystup uhlova rychlosť
Serial.print(", ");
Serial.println(u);        // Akcny zasah

```

V kóde sú matice dátové typy definované v knižnici *BasicLinearAlgebra* [].

Na Obr. 4.10 vidíme regulačný pochod LQ regulátora bez a s použitím integračnej zložky. Vidíme, že integračnú zložku sme boli nútene použiť na vynulovanie trvalej regulačnej odchýlky.



Obr. 4.10: LQ riadenie bez integračnej zložky a s integračnou zložkou.

## 4.5 Didaktické úlohy

V tejto podkapitole sformulujeme didaktické úlohy pre výučbu modelovania, identifikácie a riadenia systémov s použitím prístroja MotoShield. Každý príklad bude obsahovať definíciu zadania a návod na vypracovanie.

### 4.5.1 Modelovanie

**Príklad 4.1.** Na základe matematicko-fyzikálnej analýzy odvodte matematický model sústavy jednosmerného motora s komutátorom. Pri odvodzovaní vychádzajte z druhého Newtonovho zákona pre rotačný pohyb a druhého Kirchhoffovho zákona. Odvodené modely predstavte v spojitom a v diskrétnom čase.

Postup riešenia:

- (a) vysvetlite a uveďte pohybovú rovnicu pre jednosmerný motor –  $J\vec{\omega} = \sum_{x=1}^n \vec{M}_x$ ,
- (b) vysvetlite a uveďte rovnicu úbytku napätia na motore –  $\sum_{k=1}^n U_k = 0$ ,
- (c) vyjadrite diferenciálne rovnice v explicitnom tvare  $f^{(n)} = g(f', f'', \dots, f^{(n)})$ ,
- (d) pomocou Laplaceovej transformácie prevedte rovnice do obrazovej oblasti,
- (e) pomocou substitúcie vyjadrite prenosovú funkciu  $G(s)$ , kde je výstupom uhlová rýchlosť  $\omega(s)$  a vstupom je napätie  $U(s)$ .
- (f) vyjadrite diferenciálne rovnice v diskrétnom čase pomocou Eulerovej metódy. Zvoľte si ľubovoľnú periódu vzorkovania  $T_s$  z intervalu  $[0.01, 0.2]$  sekúnd,
- (g) stanovte Z-obraz prenosovej funkcie pomocou Z-transformácie<sup>4</sup>,
- (h) parametre systému si zvoľte ľubovoľne.
- (i) vyšetrite stabilitu systému na základe charakteristického polynómu a zmeňte parametre tak, aby obidva prenosy boli stabilné.

Vykonalte numerickú simuláciu v prostredí MATLAB a porovnajte priebeh odozvy prenosovej funkcie v obrazovej oblasti a v Z-oblasti. Porovnajte odozvu pri zmene vzorkovacej periódy  $T_s$ .

Orientačné hodnoty parametrov:

$$\begin{aligned} J &= 0.01 \text{ kg m}^2, \\ b &= 0.1 \text{ N m s}, \\ K_e &= 0.01 \text{ V rad}^{-1} \text{ s}^{-1}, \\ K_t &= 0.01 \text{ N m A}^{-1}, \\ R &= 1 \Omega, \\ L &= 0.5 \text{ H}. \end{aligned}$$

□

---

<sup>4</sup>Výsledok si overte pomocou príkazu `c2d` v MATLAB-e.

**Príklad 4.2.** Na základe matematicko-fyzikálnej analýzy odvodte matematický model sústavy jednosmerného motora s komutátorom. Pri odvodzovaní vychádzajte z druhého Newtonovho zákona pre rotačný pohyb a druhého Kirchhoffovho zákona. Odvodené modely predstavte v spojitom a v diskrétnom čase.

Postup riešenia:

- (a) vysvetlite a uveďte pohybovú rovnicu pre jednosmerný motor –  $J\vec{\omega} = \sum_{x=1}^n \vec{M}_x$ ,
- (b) vysvetlite a uveďte rovnicu úbytku napäťia na motore –  $\sum_{k=1}^n U_k = 0$ ,
- (c) vyjadrite diferenciálne rovnice v explicitnom tvare  $f^{(n)} = g(f', f'', \dots, f^{(n)})$ ,
- (d) diferenciálne rovnice zapíšte v podobe stavovo-priestorovej reprezentácie, kde za prvý stav považujeme uhlovú rýchlosť  $\omega(t)$  a druhý stav je el. prúd  $i(t)$ . Na systéme meriame obidva stavy a matica priamej väzby vstupu na výstupy  $\mathbf{D}$  je nulová.
- (e) stavový model diskretizujte pomocou Eulerovej metódy so vzorkovaciou periódou z intervalu  $[0.01, 0.2]$  sekúnd.
- (f) parametre modelu zvoľte ľubovoľne,
- (g) vyšetrite stabilitu systémov analýzou matice dynamiky a zmeňte hodnoty parametrov tak, aby obidva modely boli stabilné,
- (h) zobrazíte póly modelu v komplexnej rovine,

Vykonalajte numerickú simuláciu v prostredí MATLAB a porovnajte priebeh odozvy spojitého modelu a diskrétneho. Porovnajte odozvu pri zmene vzorkovacej periódy  $T_s$ .

Orientačné hodnoty parametrov:

$$\begin{aligned} J &= 0.01 \text{ kg m}^2, \\ b &= 0.1 \text{ N m s}, \\ K_e &= 0.01 \text{ V rad}^{-1} \text{ s}^{-1}, \\ K_t &= 0.01 \text{ N m A}^{-1}, \\ R &= 1 \Omega, \\ L &= 0.5 \text{ H}. \end{aligned}$$

□

### 4.5.2 Identifikácia modelu

**Príklad 4.3.** Odhadnite neznáme parametre systému pomocou identifikácie grey-box modelu prenosovej funkcie jednosmerného motora na prístroji MotoShield. Identifikovaný model diskretizujte a verifikujte.

Postup riešenia:

- (a) na základe dole uvedených diferenciálnych rovníc, pomocou Laplaceovej transformácie vyjadrite prenosovú funkciu motora, kde výstupom je uhlová rýchlosť  $\omega(s)$  a vstupom je napätie  $U(s)$ .
- (b) Navrhnite testovací signál pre vykonanie experimentu na zber dát<sup>5</sup>. Je odporúčané aby signál bol viachladinový s minimálnou frekvenciou zmeny amplitúdy 0.25 Hz. Horná hranica amplitúdy signálu je 5 V a dolnú si určte sám.
- (c) Napíšte program na zber dát. Použite dostupné programátorské rozhrania pre prístroj MotoShield — (Arduino IDE<sup>6</sup>, MATLAB)<sup>7</sup> alebo Simulink. Na identifikáciu budete potrebovať hodnotu vstupu a hodnotu výstupu v základných jednotkách. Je dôležité dáta boli merané vzorkovane.
- (d) Odhadnite správnu periódu vzorkovania na základe dynamiky systému.
- (e) V prostredí MATLAB si vytvorte pre dátá na identifikáciu príkazom `iddata`.
- (f) Definujte začiatočný odhad hodnoty parametrov. Odpor motora R a indukčnosť cievky L zmerajte pomocou RLC metra.
- (g) Vytvorte iniciálny model s odhadnutými hodnotami parametrov na identifikáciu príkazom `idtf`. Pozrite si dokumentáciu príkazu na úpravu parametrov objektu (minimálna, maximálna hodnota parametra a pod.).
- (h) Pomocou príkazu `tfest` vykonajte identifikáciu modelu.
- (i) Model zdiskretizujte a vyšetrite stabilitu identifikovaných modelov.
- (j) Pomocou príkazov `compare` alebo `sim` urobte validáciu identifikovaného modelu.

Model systému:

$$\begin{aligned} J\ddot{\theta} &= M_h - M_t = K_t i - b\dot{\theta} \\ U &= Ri + L \frac{di}{dt} + K_e \dot{\theta} \end{aligned}$$

□

---

<sup>5</sup>K dispozícii je funkcia `aprbsGenerate` v rámci projektu AutomationShield.

<sup>6</sup>Pre prostredie Arduino IDE na vzorkovanie použite premennú `MotoShield.stepEnable` — nadobudne kladnú hodnotu na konci každej vzorky.

<sup>7</sup>Použite metódy `actuatorWriteVolt` a `sensorReadRadian`.

**Príklad 4.4.** Odhadnite neznáme parametre systému pomocou identifikácie grey-box stavového modelu jednosmerného motora na prístroji MotoShield. Identifikovaný model diskretizujte a verifikujte.

Postup riešenia:

- na základe dole uvedených diferenciálnych rovníc vyjadrite stavový model motora, kde prvým stavom je uhlová rýchlosť  $\omega(t)$ , druhým stavom je prúd  $i(t)$  a vstupom je napätie  $U(t)$ . Na prístroji meriame obidva stavy a matica priamej väzby vstupu na výstup je nulová.
- Navrhnite testovací signál pre vykonanie experimentu na zber dát<sup>8</sup>. Je odporúčané aby signál bol viachladinový s minimálnou frekvenciou zmeny amplitúdy 0.25 Hz. Horná hranica amplitúdy signálu je 5 V a dolnú si určte sám.
- Napíšte program na zber dát. Použite dostupné programátorské rozhrania pre prístroj MotoShield — (Arduino IDE<sup>9</sup>, MATLAB)<sup>10</sup> alebo Simulink. Na identifikáciu budete potrebovať hodnotu vstupu a hodnotu výstupu v základných jednotkách. Je dôležité dáta boli merané vzorkovane.
- Odhadnite správnu periódu vzorkovania na základe dynamiky systému.
- V prostredí MATLAB si vytvorte pre dátá na identifikáciu príkazom `iddata`.
- Definujte začiatočný odhad hodnoty parametrov. Odpor motora R a indukčnosť cievky L zmerajte pomocou RLC metra.
- Vytvorte iniciálny model so začiatočným odhadom parametrov na identifikáciu príkazom `idss`. Pozrite si dokumentáciu príkazu na úpravu parametrov objektu (minimálna, maximálna hodnota parametra a pod.).
- Pomocou príkazu `ssest` identifikujte model.
- Model diskretizujte a vyšetrite stabilitu identifikovaných modelov.
- Pomocou príkazov `compare` alebo `sim` urobte validáciu identifikovaného modelu.

Model systému:

$$\begin{aligned} J\ddot{\theta} &= M_h - M_t = K_t i - b\dot{\theta} \\ U &= Ri + L \frac{di}{dt} + K_e \dot{\theta} \end{aligned}$$

□

---

<sup>8</sup>K dispozícii je funkcia `aprbsGenerate` v rámci projektu AutomationShield.

<sup>9</sup>Pre prostredie Arduino IDE na vzorkovanie použite premennú `MotoShield.stepEnable` — nadobudne kladnú hodnotu na konci každej vzorky.

<sup>10</sup>Použite metódy `actuatorWriteVolt`, `sensorReadCurrent` a `sensorReadRadian`.

### 4.5.3 Riadenie

**Príklad 4.5.** Navrhnite vstupno-výstupné regulátory pre systém jednosmerného motora na prístroji MotoShield. Riadime uhlovú rýchlosť motora  $\omega$ . V postupe riešenia sú uvedené typy vstupno-výstupných regulátorov, ktoré nalaďte heuristicky a potom aj pomocou ľubovoľnej metódy ladenia na základe diskrétnej prenosovej funkcie.

Postup riešenia:

- (a) identifikujte diskrétnu prenosovú funkciu jednosmerného motora alebo použite prenosovú funkciu z príkladu 4.3<sup>11</sup>.
- (b) Určte správnu hodnotu periódy vzorkovania  $T_s$  na základe dynamiky systému.
- (c) Navrhnite regulačné obvody s P, PI a PID regulátormi metódou heuristiky (pokus-omyl). Pred vyskúšaním regulácie na hardvéri pomocou príkazu `step`, nasimulujte odozvu regulačných obvodov na jednotkový skok. Prípadne zmeňte konštanty regulátora tak, aby odozva regulačných obvodov bola stabilná.
- (d) Zvoľte si prostredie v ktorom príklad vypracujete — použite dostupné programátorské rozhranie pre prístroj MotoShield (Arduino IDE, MATLAB alebo Simulink). Zadefinujte hodnotu periódy, vektor žiadanej hodnoty a konštanty regulátora.
- (e) Vytvorte cyklus v ktorom sa bude vykonávať riadiaci algoritmus<sup>1213</sup>. Spustite program regulácie na hardvéri a vykreslite výsledky regulátorov do jedného grafu v porovnaní so žiadanou hodnotou<sup>14</sup>.
- (f) Navrhnite konštanty regulátora pomocou jednej s metódou ladenia PID regulátorov na základe prenosovej funkcie. Vykonajte regulačiu a porovnajte výsledky rovnakých regulátorov (každý do osobitného obrázku).
- (g) Popíšte výsledky regulácie a vplyv jednotlivých zložiek P, I a D na regulačný pochod.
- (h) Na základe regulačnej odchýlky vypočítajte indikátor kvality riadenia pre regulačné pochody s PID riadením.

□

---

<sup>11</sup>Dbajte na to aké sú jednoty vstupnej a výstupnej veličiny prenosovej funkcie. Použite rovnaké jednotky a vzorkovaciu periódu pre každý regulátor!

<sup>12</sup>V prostredí Arduino IDE použite premennú `MotoShield.stepEnable` — nadobúda kladnú hodnotu na konci každej vzorky

<sup>13</sup>V prostredí MATLAB použite príkazy `tic` a `toc`.

<sup>14</sup>Ziadanú hodnotu si zvoľte ľubovoľnú — inšpirujte sa Obr.4.11 a 4.12.

**Príklad 4.6.** Navrhnite stavový regulátor pre systém jednosmerného motora na prístroji MotoShield. Riadime uhlovú rýchlosť motora  $\omega$ . V postupe riešenia sú uvedené typy stavových regulátorov, ktoré naladíte riešením optimalizačnej úlohy na základe stavového modelu.

Postup riešenia:

- (a) identifikujte diskrétny stavový model jednosmerného motora alebo použite model z príkladu 4.4<sup>15</sup>.
- (b) Určte správnu hodnotu periódy vzorkovania  $T_s$  na základe dynamiky systému.
- (c) Vyšetrite stabilitu a riadiťenosť stavového modelu.
- (d) Uveďte riadiaci zákon LQ regulátora pri nenulovej žiadanej hodnote.
- (e) Navrhnite penalizačné matice  $\mathbf{Q}$  a  $\mathbf{R}$  LQ regulátora a vypočítajte optimálne zosilnenie  $\mathbf{K}$ .
- (f) Vyšetrite stabilitu regulačného obvodu na základe vlastných čísel. Použite príkaz `eig`. V prípade, že je obvod nestabilný, upravte váhovanie stavov a vstupu tak aby obvod bol stabilný.
- (g) Zvoľte si prostredie v ktorom príklad vypracujete — použite dostupné programátorské rozhranie pre prístroj MotoShield (Arduino IDE, MATLAB alebo Simulink). Zadefinujte hodnotu periódy, vektor žiadanej hodnoty a konštanty regulátora.
- (h) Vytvorte cyklus v ktorom sa bude vykonávať riadiaci algoritmus a odhad druhého stavu pomocou Kalmanovho filtra<sup>16</sup><sup>17</sup>. Spustite program regulácie na hardvéri a vykreslite výsledky regulátorov do jedného grafu v porovnaní so židanou hodnotou<sup>18</sup>.
- (i) Vykonalte reguláciu a zobrazte regulačný pochod.
- (j) Zakomponujte integračnú zložku do LQ regulátora a porovnajte výsledky s LQ bez integračnej zložky. Zopakujte krok (e) a (f).
- (k) Popíšte výsledky regulácie a integračnej zložky pri riadení.

□

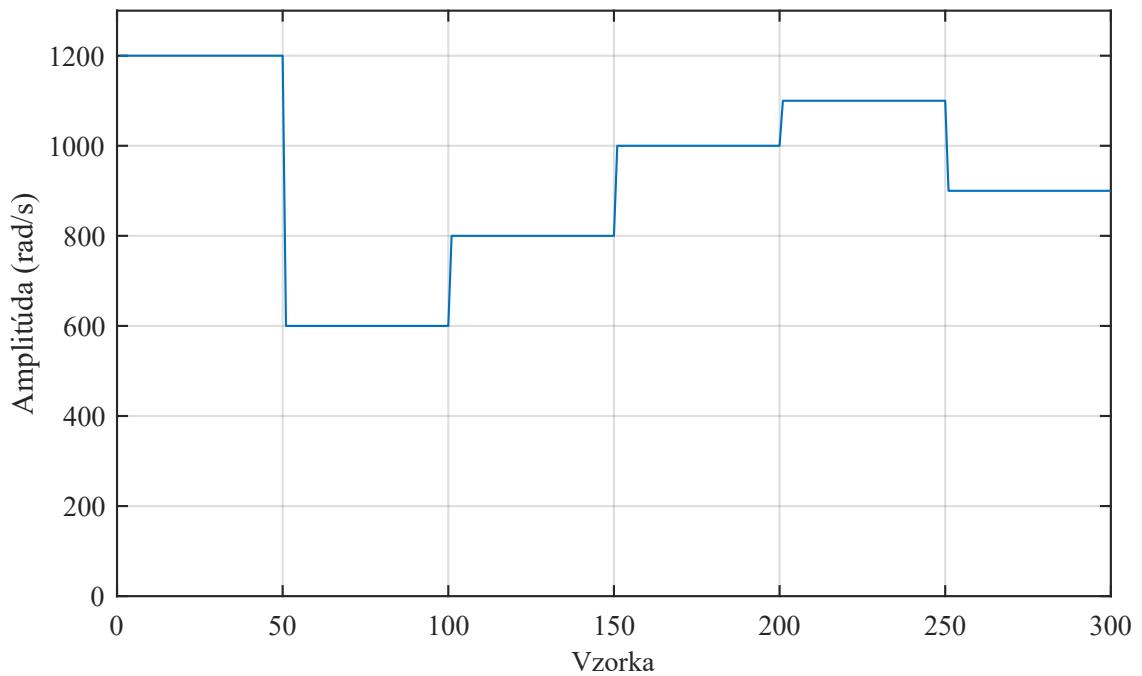
---

<sup>15</sup>Používajte základné jednotky pre každú veličinu.

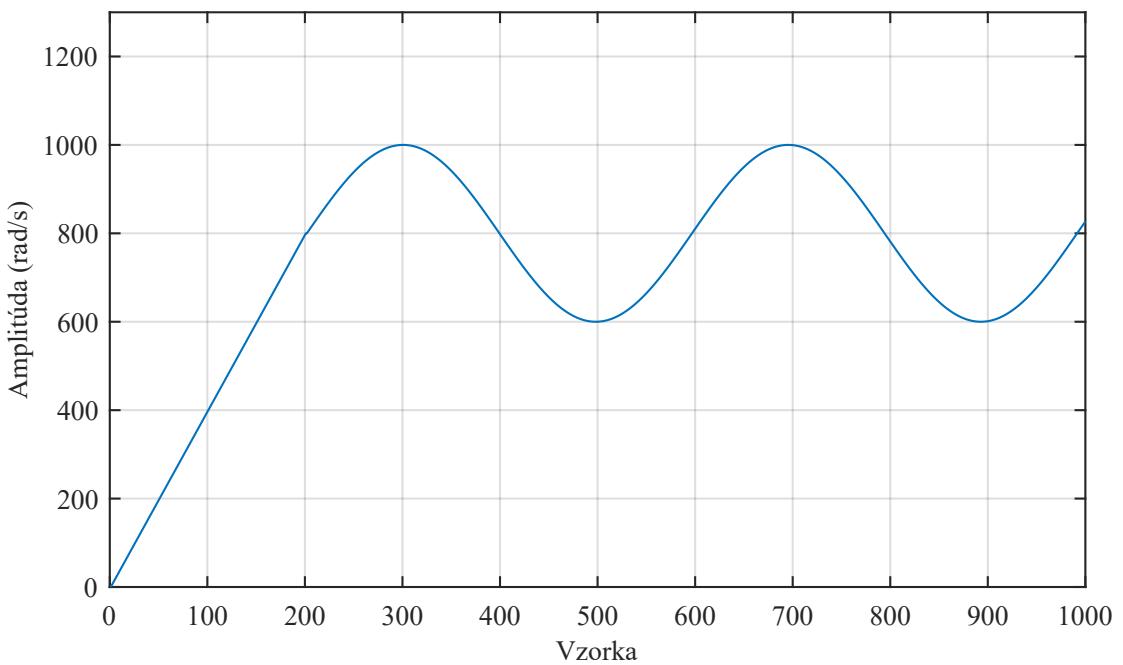
<sup>16</sup>V prostredí Arduino IDE použite premennú `MotoShield.stepEnable` — nadobúda kladnú hodnotu na konci každej vzorky

<sup>17</sup>V prostredí MATLAB použite príkazy `tic` a `toc`.

<sup>18</sup>Zidanú hodnotu si zvoľte ľubovoľnú — inšpirujte sa Obr.4.11 a 4.12.



Obr. 4.11: Viac hladinový signál žiadanej hodnoty.



Obr. 4.12: Sínusový signál žiadanej hodnoty.

## 5 Grafické užívateľské rozhranie

Základnou ideou tvorby grafického užívateľského rozhrania pre projekt *AutomationShield* je poskytnúť možnosť na spustenie vopred prichystaných programov, bez toho aby sa užívateľ musel vynachádzať v niektorom z podporovaných programovacích jazykov. Motiváciou tvorby takéhoto rozhrania je prilákať ľudí, ktorí očakávajú a preferujú „out-of-box“ skúsenosť.

Grafické rozhranie bolo koncipované, tak aby splňalo nasledovné funkcionality:

- nahrať vopred vytvorené príklady<sup>1</sup> na mikroradičovú dosku, napr. na PID riadenie.
- Poskytnúť možnosť užívateľovi zmenu parametrov.
- Čítať dátu z mikroradičovej doske cez sériový port.
- Zobraziť vyčítané dátu.

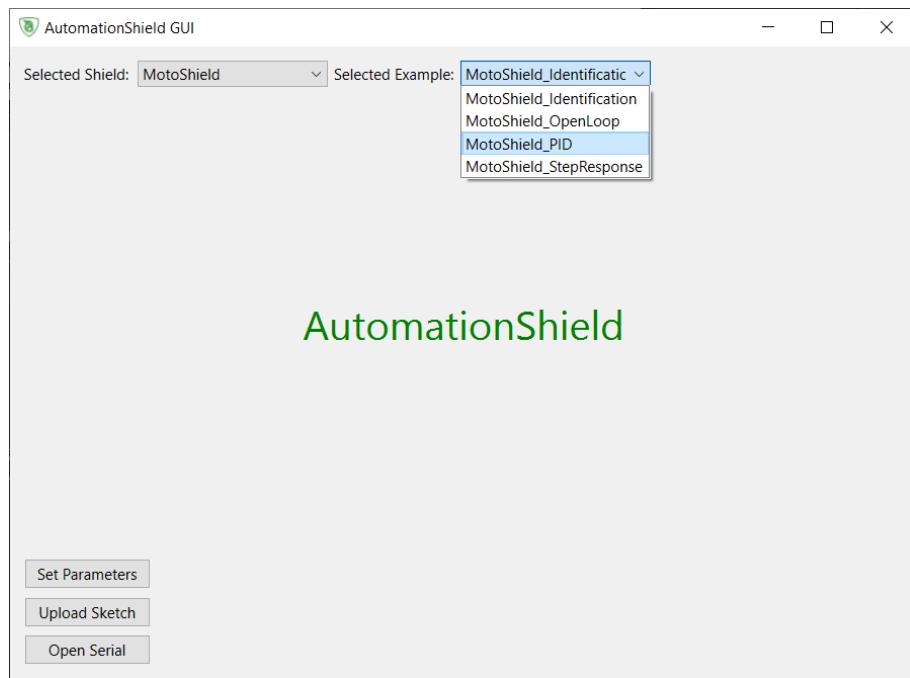
Na tvorbu grafického užívateľského rozhrania existuje množstvo knižníc (angl. Framework). Pri voľbe knižnice sme rozhodovali hlavne na základe parametrov ako je popularita, kvalita dokumentácie a do konca aj úspech na komerčnom trhu. Jedna z najzastúpenejších knižníc je *Qt*. Jej výhodou je, že podporuje niekoľko programovacích jazykov (JS, C++ a python), beží na viacerých platformách a široko je používaná v priemysle. Hlavne sa používa na tvorbu grafického rozhrania v automobilovom priemysle, v zdravotníctve ale dokonca aj v priemyselnej automatizácii. Knižnica je napísaná v jazyku C++ a taktiež je najviac aj používaná v tomto jazyku. Hlavne preto, že sa často používa na vnorených systémoch (angl. Embedded Systems) [21].

Na vývoj aplikácie sme si zvolili programovací jazyk python. Pri tomto rozhodnutí sme sa hlavne riadili podporou knižníc na rôzne účely ako napr. spracovanie a vyhľadávanie súborov, operácie s rôznymi dátovými typmi a grafické zobrazovanie dát. Taktiež, niektoré koncepty ako napr. odvoz odpadu (angl. Garbage Collection) sú v programovacom jazyku python automatizované [15], na rozdiel od jazyku C++. Toto je veľkou výhodou hlavne pre začiatočníkov zvlášť pri riešení komplexnejších úloh.

Postup, ktorým sa jadro grafického rozhrania riadi popíšeme v tomto odstavci. Pri spustení grafického rozhrania je užívateľ vyzvaný aby uviedol cestu k priečinku repozitára *AutomationShield*. Toto je umožnené príkazom `QFileDialog`. Program si cestu uloží do premennej. Následne sa otvorí okno grafického rozhrania, viď

---

<sup>1</sup>Myslí sa na príklady, ktoré sa nachádzajú v repozitáre *AutomationShield*.



Obr. 5.1: Grafické užívateľské rozhranie.

Obr. 5. Kedže budeme pracovať s napísanými príkladmi pre prostredie Arduino IDE, cesta sa presmeruje do priečinku „*AutomationShield/examples*“. Tu sa nachádzajú súbory pre každý z prístrojov. Súbory v tomto priečinku sa prepíšu a zadefinujú sa ako možnosti pre objekt padajúceho menu `shieldMenu` s popisom „Selected Shield“:. Padajúce menu je typu `QComboBox`. Akonáhle si užívateľ zvolí niektorý z prístrojov, spustí sa metóda `onShieldChanged`, ktorá presmeruje cestu do vnútra priečinku príkladov zvoleného prístroja. Táto metóda zmení hodnoty padajúceho menu `exampleMenu`. Toto menu má funkciu voľby jedného z dostupných príkladov pre zvolený prístroj. Na Obr. 5 má popis „Selected Example“:. Kedž si užívateľ zvolí jeden z príkladov, spustí sa metóda `onExampleChanged`. Táto metóda načítava konfigurovateľné parametre zvoleného príkladu, zavolaním metódy `getParameters`, ktorá bola napísaná v rámci triedy `exampleHandler`. Metóda `getParameters` je veľmi dôležitá, kedže nesie informácie o tom, ktoré z parametrov sú konfigurovateľné pre každý príklad. Preto je napísaná v rámci osobitnej triedy, resp. osobitného súboru. V rámci tejto metódy sú parametre pre jednotlivé príklady zapisované nasledovne:

```
class ExampleHandler():
    @staticmethod
    def getParameters(example):
        match example:
            case 'MotoShield_Identification':
                return ["TS", "PRBS"]
            case 'MotoShield_PID':
                return ["TS", "KP", "TI", "TD"]\text{.}
```

Vidíme, že pre príklad `MotoShield_Identification` sú nastaviteľné parametre vzorkovacia periód TS a typ signálu PRBS. Pre príklad `MotoShield_PID`

sú nastaviteľné parametre vzorkovacia períoda TS a konštanty PID regulátora KP, TI a TD.

V rámci metódy `onExampleChanged` je zvolaná aj metóda na vygenerovanie vstupných polí pre nastavenie parametrov `generateInputFields`. Táto metóda v rámci cyklu vytvorí objekty typu `QLineEdit` a `QLabel`. Vytvorené objekty vstupných polí sú pomenované podľa názvu daného parametru parametrov, kým objekty popisov majú ešte príponu `Label`. Cyklus vyzerá nasledovne:

```
for i in range(len(parameters)):
    globals()[parameters[i] + "Label"] = QtWidgets.QLabel(parameters[i])
    self.layoutInputFields.addWidget(globals()[parameters[i] + "Label"])
    globals()[parameters[i]] = QtWidgets.QLineEdit()
    self.layoutInputFields.addWidget(globals()[parameters[i]])
    globals()[parameters[i]].setFixedWidth(70)
```

Ked' už máme zadané hodnoty parametrov vo vstupných poliach, posunieme sa na ich prepis do príkladu. Ked'že nechceme zasahovať priamo do súboru napísaného príkladu, použijeme pomocný, dočasný súbor `Upload_File.ino`. Po stlačení tlačidla „Set Parameters“ sa vykoná metóda `copyAndReplace`. Táto metóda je jedna z komplexnejších v rámci tohto súboru. Jej úlohou je najprv skopírovať celý obsah zo súboru zvoleného príkladu do dočasného súboru. Následne v rámci cyklu vyhľadá parametre, ktorých hodnoty chceme meniť a pomocou regulárneho výrazu (angl. Regular Expression<sup>2</sup>) zmeníme prednastavené hodnoty za hodnoty uvedené v textových poliach. Tlačidlo „Set Parameters“ taktiež zavolá aj metódu na vyčítanie rýchlosťi sériovej komunikácie `getSerialBaudrate`. Toto je tiež vykonané pomocou regulárneho výrazu, ked'že je v každom príklade táto informácia zapísaná ako argument príkazu `Serial.begin()`. Príkaz nevyčítanie tejto hodnoty je nasledovný:

```
self.baudRate = re.findall(r"Serial.begin\((\d*)\);", filedata)[0] \text{,}
```

kde premenná `filedata` obsahuje kompletný príklad vo forme reťazca. Informáciu o rýchlosti prenosu dát budeme potrebovať na čítanie dát zo sériového portu.

Na nahranie súboru budeme použiť externý program Arduino CLI (angl. Command-Line Interface<sup>3</sup>). Arduino CLI je oficiálny softvér z projektu Arduino. Poskytuje nám rovnaké funkcionality ako vývojové prostredie Arduino IDE, vykonávať prostredníctvom príkazového riadku. V jazyku python sa vieme na úroveň príkazového riadku dostať cez knižnicu `os`. Príkazy spúšťame pomocou metódy `os.system()`, kde príkaz uvedieme do argumentu v tvare reťazca.

Stlačením tlačidla „Upload Sketch“ nahráme dočasne vytvorený súbor na dosku. Sekvencia príkazou, ktoré nám toto umožnia je zahrnutá v metóde `uploadToBoard`. Najprv musíme zistiť, či je doska vôbec pripojená na sériovom porte a o akom type dosky sa jedná. Toto vykonáme cli príkazom `arduino-cli board list`. Tento príkaz vráti množstvo informácií, z ktorých potrebujeme iba vedieť číslo sériového portu cez ktorý je pripojená a typ dosky (FQNB). Ked' zistíme tieto informácie, príklad skompilujeme a potom nahráme na mikroradičovú dosku. Kompilácia a nahratie koná posledný príkaz metódy `uploadToBoard`, kde pomocou operátoru `{}` a metódy `format` vnášame premenné do reťazca.

<sup>2</sup>abbrev. RegEx.

<sup>3</sup>slov. Príkazový riadok.

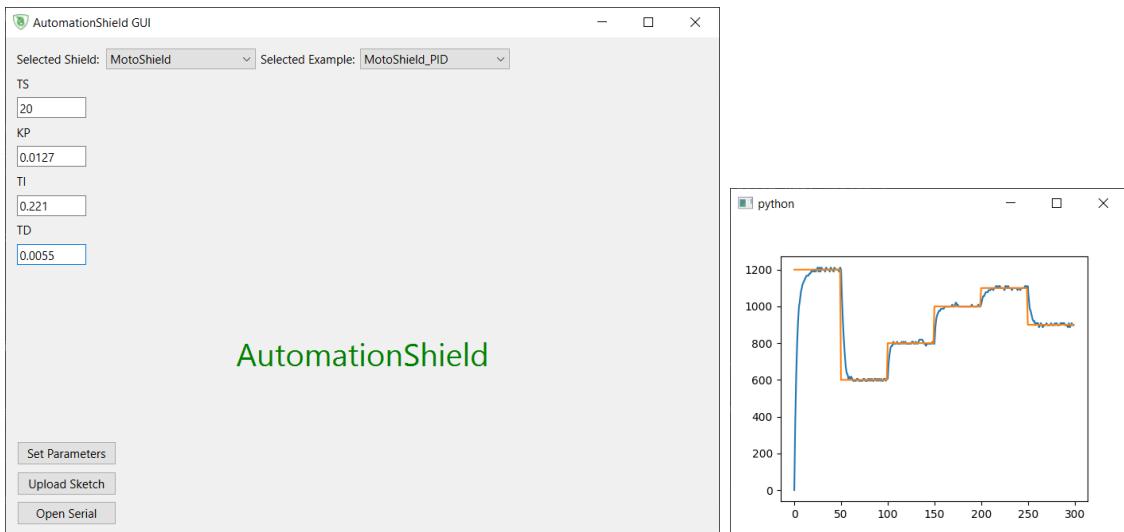
```

def uploadToBoard(self):
    self.boardScan=subprocess.getoutput('arduino-cli board list')
    self.comPort=self.boardScan.split()[7] # Read COM Port Number
    self.boardType=self.boardScan.split()[14] # Read FQBN - Board type
    os.system("arduino-cli compile --upload {} --port {} --fqbn
    {}".format(self.dir + "\\\"+self.shieldMenu.currentText() +
    "\\\"+self.exampleMenu.currentText() +
    "\\\"+self.exampleMenu.currentText() +
    ".ino",self.comPort,self.boardType))

```

Na koniec, nám zostáva iba vyčítať hodnoty, ktoré nám mikroradič späť posielá. Na čítanie zo sériového portu použijeme knižnicu `Serial`. Čítanie dát zo sériového portu sa začne až po stlačení tlačidla „Open Serial“. V tom okamihu sa spustí funkcia metóda `monitorFromBoard` v rámci ktorej sa vytvorí objekt na sériovú komunikáciu a otvorí sa prietok dát. V prípade, že mikroradič žiadne dátá nevysielá, posielané budú prázdne bajty (inými slovami nebude žiadne napätie žilach káblu). Rovnakoý prípad sa stane aj keď program na mikroradiči dobehne. Preto potrebujeme pomocnú premennú `flag`, na to aby sme rozpoznali či sa program na mikroradiči už ukončil alebo ani nezačal. Prednastavená hodnota premennej je záporná a až, keď sa v kábli prejaví napätie, premená nadobudne kladnú hodnotu. V prípade, že je premenná `flag` kladná a zároveň nám prichádzajú prázdne bajty, sériová komunikácia sa ukončí. Čítanie zo sériového portu je vykonávané v kvázi nekonečnom cykluse bez prerušení, s cieľom aby sa dátá vyčítali bez porúch.

Na koniec, načítané dátá predformátujeme a vykreslíme ich pomocou knižnice `matplotlib`. Načítané dátá sa vykreslia až keď program na mikroradiči dobehne.



(a) Grafické rozhranie

(b) Okno zobrazenia dát

Obr. 5.2: PID riadenie cez grafické rozhranie.

Chápeme, že rozhranie nie je dokonalé. Medzi najzávažnejšie nedostatky patria:

- manažment a hlásenie chýb (angl. Error Handling). Rozhranie neindikuje chyby pri kompliacii Arduino programu, ani chyby zle špecifikovaného priečinku re-

pozitára. Taktiež, program nereaguje, keď užívateľ namiesto čísel uvedie písaná do vstupných polí. V takom prípade sa program Arduino neskompiluje ale užívateľ nemá spätnú informáciu.

- Dáta nie sú vykreslované v kvázi reálnom čase, ale až po ukončení sériovej komunikácie. Toto riešenie je iba dočasné<sup>4</sup>.
- Dáta z experimentu nevieme zatiaľ exportovať, čo môže byť problematické kebyže chceme ich ďalej spracovať.
- Dáta sa dajú iba vykreslovať. Bolo by praktické (na diagnostiku) kebyže sa hodnoty dát vypisujú v ukotvenom okne grafického rozhrania.
- Lokálny repozitár nie je porovnávaný pomocou prístroja *git*. Po spustení aplikácie by sa mal porovnať lokálny priečinok s repozitárom. Takto by sa poskytla možnosť aktualizácie kódu.
- Z estetickej stránky je veľa nedostatkov. Prvotná vec, keď ide o estetiku, je zabezpečiť responzívny dizajn.

---

<sup>4</sup>V rámci programu *AutomationShield* [9] je softvér priebežne vyvíjaný.

## 6 Záver

Zámerom tejto práce bol vývoj didaktického prístroja na riadenie jednosmerného motora s komutátorom. Na vývoj takého zariadenia sme sa museli bližšie zoznámiť s problematikou modelovania, identifikácie a riadenia spomenutého systému.

Pri návrhu druhej verzie prístroja MotoShield sme sa snažili eliminovať nedostatky prvej verzie prístroja. Hardvér sme vylepšili po stránke merania výstupných veličín. Pri meraní uhlovej rýchlosťi sme správnym zapojením druhého kanálu Hallovho snímača dostali dva-krát viac impulzov za jednotku času, čo umožnilo zmeniť vzorkovaciu períodu. Toto je veľmi dôležité najmä pri riadení. Na meranie prúdu sme vylepšili perifériu, čo umožnilo kvalitnejšie identifikovať stavový model systému. V rámci projektu je preferované, aby prístroje boli kompatibilné s mikroradičovými doskami SAM a SAMD architektúry. Toto pravidlo dizajnu sme dodržali a úspešne sme otestovali prístroj na doske Arduino Due. V prípade pohonu motora sme navrhli integrovaný obvod s viacerou funkcionálitou ako bola v prípade prvej verzie motora. Hoci sme túto funkcionality nevyužili, v ďalšej verzie prístroja môžeme zahrnúť AD prevodník a skúsiť pohon motora ovládať pomocou vnútorného oscilátora IC ZXBM5210.

Navrhnuté programátorské prostredia bolo úspešne otestované a prišli sme k záveru, že sú plne funkčné. V ďalšom vývoji programátorského rozhrania zostáva miesto na pridanie ďalších funkcionálít, ako napr. detekcia smeru otáčania. V Simulink-u by bolo užitočné pridať možnosť definovania smeru otáčania v podobe externého pravdivostného signálu.

Pomocou každého navrhnutého programátorského rozhrania bol napísaný jeden príklad na riadenie systému, pričom sme si v základných bodech vysvetlili analytický prístup návrhu regulátorov. Pri každom príklade bol systém úspešne uregulovaný.

Na základe prvej verzie prístroja MotoShield bol publikovaný článok pod názvom: **MotoShield: Open Miniaturized DC Motor Hardware Prototype for Control Education** [19], na vedeckej konferencii v meste Lincoln, USA 2021. V článku sa autori snažili prezentovať potenciál prístroja pre akadémiu.

Význam prístroja na didaktické účely sa osvedčil aj tento rok, keďže sa na základe MotoShield-u vypracovala bakalárska práca na Fakulte elektrotechniky a informatiky STU pod názvom: **Zostavenie a riadenie fyzikálneho modelu motorčeka ako nadstavby Arduino Uno**, kde snahou autora Šimona Nechaja bolo dozvedieť, viac o dynamike a riadení jednosmerných motorov.

# Literatúra

- [1] Arduino Reference. Defining Pin Levels: HIGH and LOW. Online, 7.2003. [cit. 25.5.2022], <https://www.arduino.cc/reference/en/language/variables/constants/constants/>.
- [2] H. Barnes, G. Bianchi, and J. Moreira. A review of 90 degree corner design for high-speed digital and mmwave applications. In *2020 IEEE 29th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, pages 1–3, 2020.
- [3] C. Belavý. *Základy automatizácie a merania*. Slovenská technická univerzita v Bratislave, Bratislava, Vazovova 5, 2012.
- [4] J. Boldocký. Motoshield: Vývoj programátorského rozhrania, identifikácia, riadenie a úprava didaktického prístroja na riadenie rýchlosťi motorov. Slovenská technická univerzita v Bratislave, 6 2020. Bakalárská práca, [cit. 25.5.2022].
- [5] T. P. Cabré, A. S. Vela, M. T. Ribes, J. M. Blanc, J. R. Pablo, and F. C. Sancho. Didactic platform for dc motor speed and position control in z-plane. *ISA Transactions*, 118:116–132, 2021.
- [6] cppreference. inline specifier. Online. [cit. 25.5.2022], <https://en.cppreference.com/w/cpp/language/inline>.
- [7] DFRobot. Micro Metal Geared motor w/Encoder. Online Store. [cit. 25.5.2022], <https://www.dfrobot.com/product-1437.html>.
- [8] DIODES Incorporated. Zxbm5210 reversible dc motor drive with speed control. Online Datasheet, 12.2013. [cit. 25.5.2022], <https://www.diodes.com/assets/Datasheets/ZXBM5210.pdf>.
- [9] G. Takács, M. Gulán. Automationshield repository. Online GitHub. [cit. 25.5.2022], <https://github.com/gergelytakacs/AutomationShield>.
- [10] J. Järvi and J. Freeman. C++ lambda expressions and closures. *Science of Computer Programming*, 75(9):762–772, 2010. Special Issue on Object-Oriented Programming Languages and Systems (OOPS 2008), A Special Track at the 23rd ACM Symposium on Applied Computing.
- [11] T. Konkoly. Experimentálne moduly pre výučbu automatizácie. Slovenská technická univerzita v Bratislave, 5 2018. Bakalárská práca, [cit. 25.5.2022].

- [12] P. Prinz and U. Kirch-Prinz. *A Complete Guide to Programming in C++*. Jones and Bartlett Publishers, Sudbury, Massachusetts, 2002.
- [13] G. Rata, C. Bejenar, and M. Rata. A solution for studying the d.c. motor control using ni myrio-1900. In *2019 8th International Conference on Modern Power Systems (MPS)*, pages 1–4, 2019.
- [14] Rick Hill,Bill Messner,Dawn Tilbury. Dc motor speed: System modeling. Online. [cit. 25.5.2022], <https://ctms.engin.umich.edu/CTMS/index.php?example=MotorSpeed&section=SystemModeling>.
- [15] A. S. Saabith, M. Fareez, and T. Vinothraj. Python current trend applications—an overview. *International Journal of Advance Engineering and Research Development*, 6(10), 2019.
- [16] G. Takács and M. Gulan. *Základy Prediktívneho Riadenia*. Spektrum STU, Bratislava, Slovakia, 1. edition, 2018. In Slovak language. (Fundamentals of Predictive Control).
- [17] G. Takács, M. Gulan, J. Bavlna, R. Köplinger, M. Kováč, E. Mikuláš, S. Zarghoon, and R. Salíni. HeatShield: a low-cost didactic device for control education simulating 3D printer heater blocks. In *Proceedings of the 2019 IEEE Global Engineering Education Conference (EDUCON)*, pages 374–383, Dubai, United Arab Emirates, April 2019.
- [18] G. Takács, J. Vachálek, and B. R. Ilkiv. Identifikácia sústav (system identification).
- [19] G. Takács, J. Boldocký, E. Mikuláš, T. Konkoly, and M. Gulan. Motoshield: Open miniaturized dc motor hardware prototype for control education. In *2021 IEEE Frontiers in Education Conference (FIE)*, pages 1–9, 2021.
- [20] Texas Instruments. Ina1x9 high-side measurement current shunt monitor. Online Datasheet, 2.2017. [cit. 25.5.2022], <https://www.ti.com/lit/ds/symlink/ina169.pdf?ts=1591356334230>.
- [21] The Qt Company. Why qt? Online. [cit. 25.5.2022], <https://www.qt.io/why-qt>.